Advanced Routing

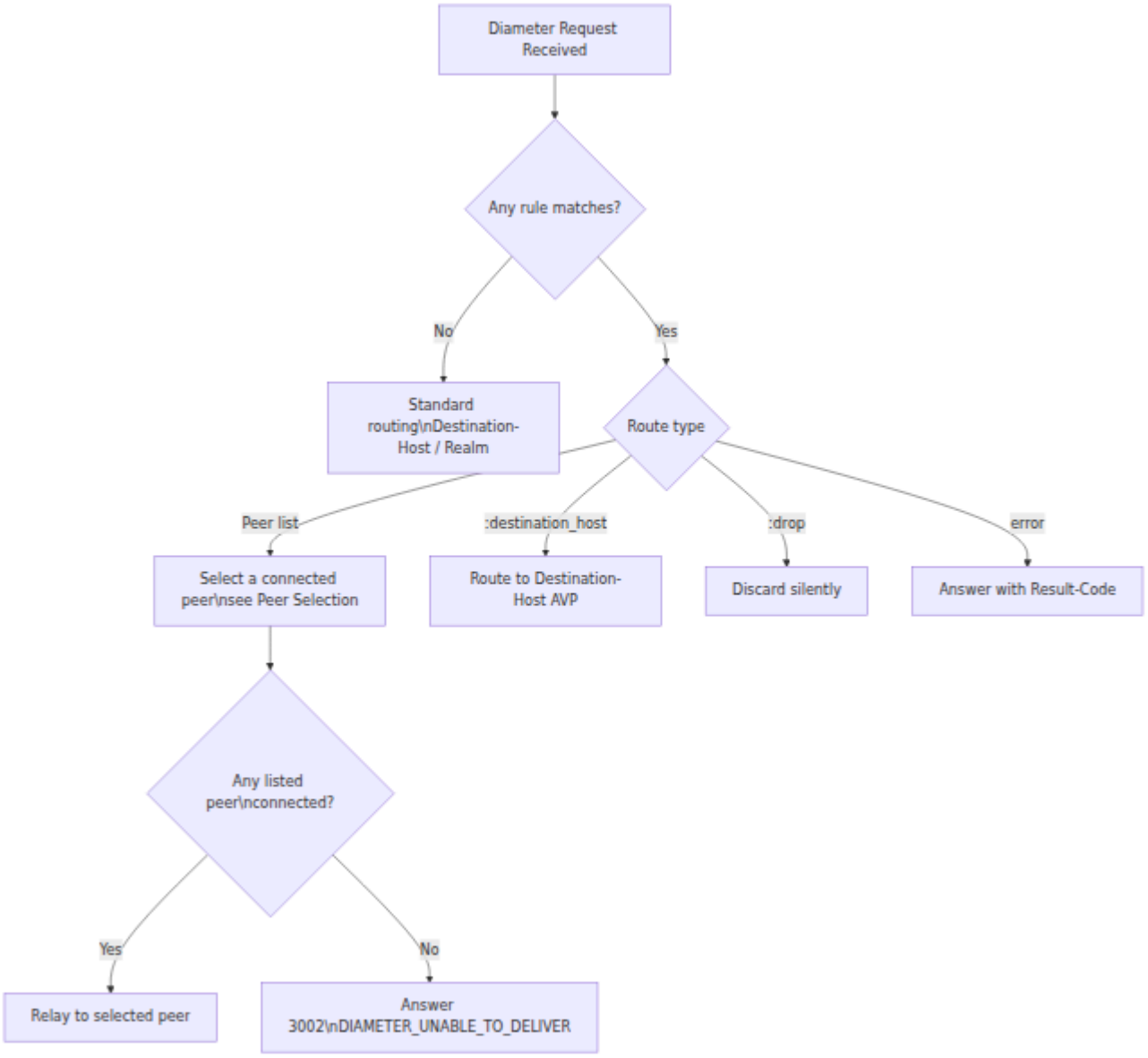
Advanced Routing lets the DRA make routing decisions based on the content of each Diameter request. It does not use **Destination-Realm** alone. **Rules** match on properties of the message, such as originating peer, Application-Id, Command-Code, and AVP values. When a rule matches, it directs the request to a chosen set of peers or a special destination.

A rule can also distribute traffic across several peers for **failover** and **load balancing**. It uses the per-peer `priority` and `weight`. For stateful applications such as Gx (PCRF) and Gy (OCS), Advanced Routing preserves **session affinity**. 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. The DRA never re-evaluates them 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

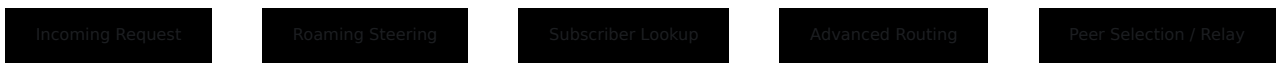
The DRA evaluates every inbound request against the configured rules **in order, from top to bottom**. The **first** rule whose filters match wins and decides the route. The DRA does not evaluate the later rules. If no rule matches, the request continues with standard **Destination-Host** / **Destination-Realm** routing.



A matched rule can resolve only to peers that are **not currently connected**. In that case, the DRA answers with **DIAMETER_UNABLE_TO_DELIVER (3002)**. The DRA never hands the request back to standard routing. Standard routing would deliver it to the wrong node and return a misleading result.

Pipeline Position

Advanced Routing runs after Roaming Steering and Subscriber Lookup. Its decision overrides the peer that standard routing would otherwise select.



Rule Processing

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

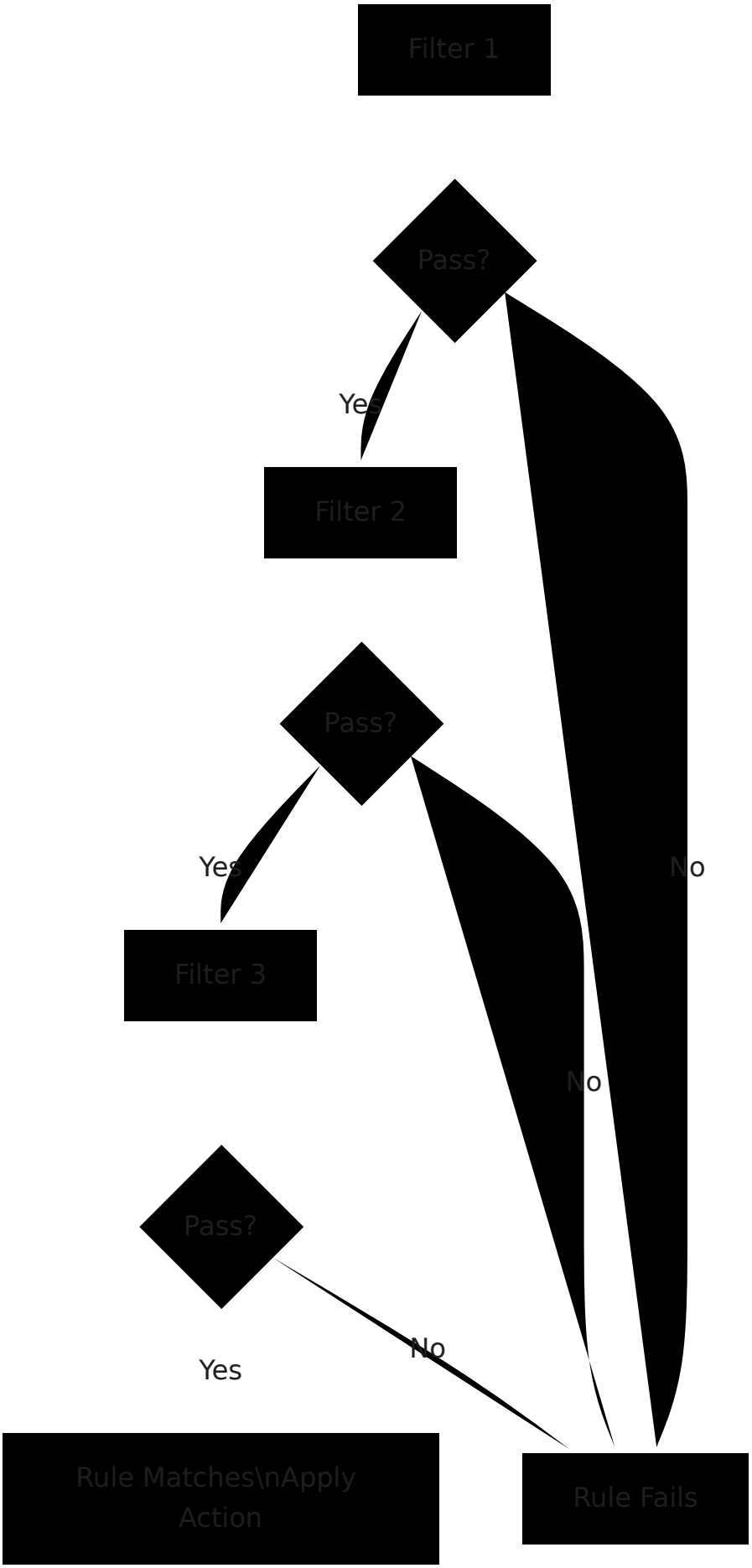
1. The DRA evaluates rules **in order from top to bottom**, as defined in the configuration.
2. The DRA combines the filters within a rule according to the rule's `match` mode (`:all`, `:any`, or `:none`).
3. The **first matching rule wins**. The DRA does not evaluate the later rules.
4. If no rule matches, the DRA uses standard routing.

Match Modes

The `match` parameter determines how the DRA combines a rule's filters.

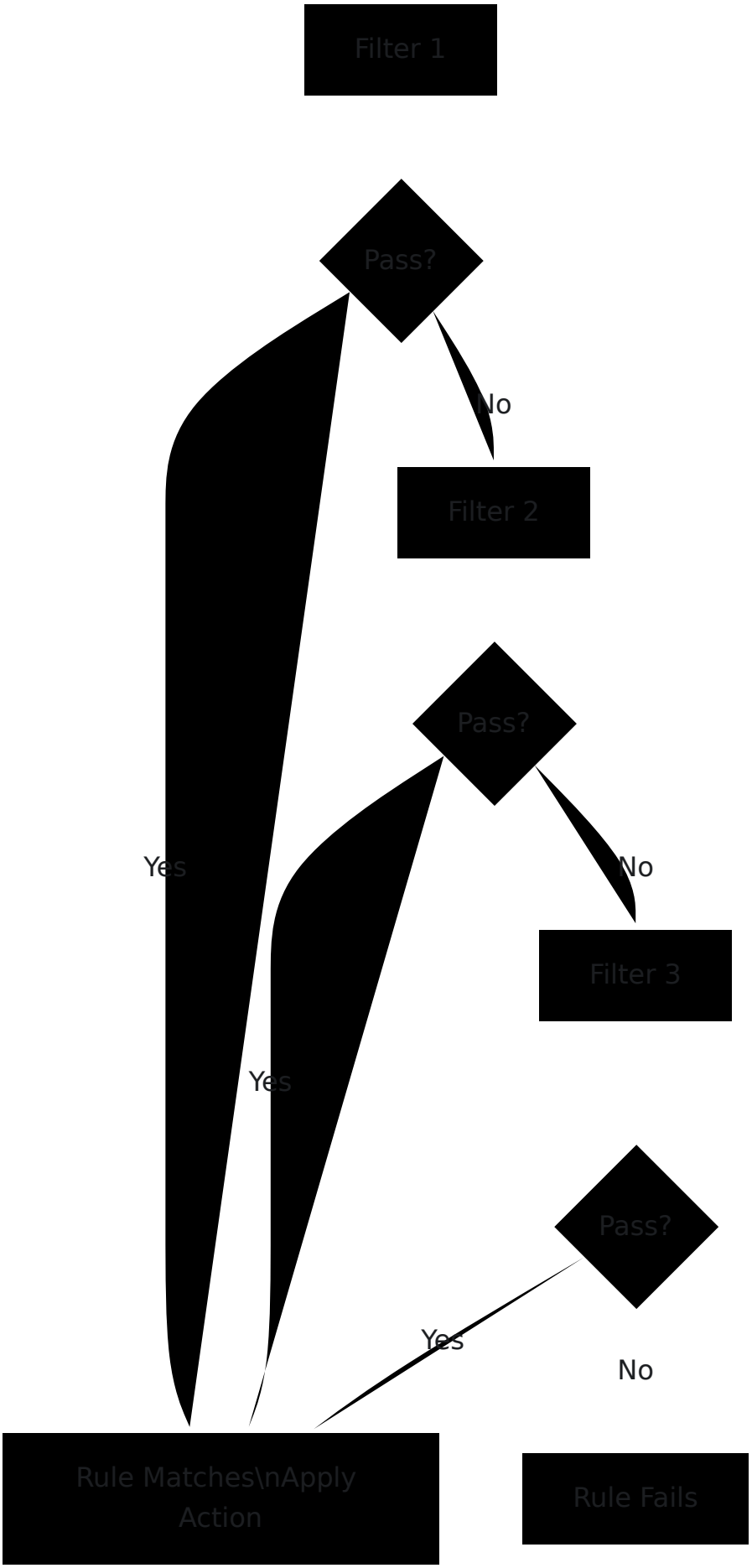
`:all` (AND logic)

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

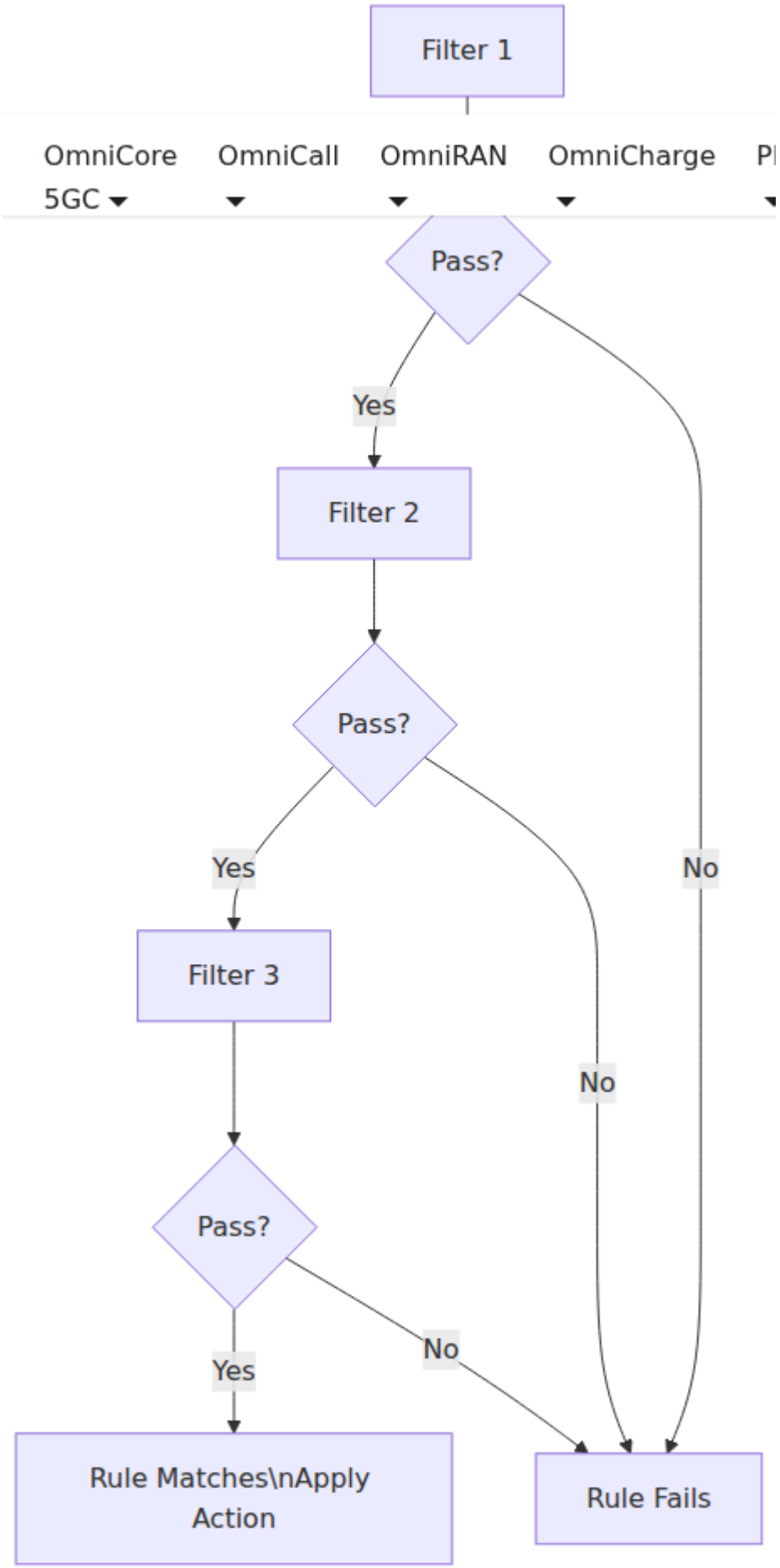
`:any` (OR logic)

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. This is inverse matching. With three filters, `NOT filter1 AND NOT filter2 AND NOT filter3` must all hold for the rule to succeed. Every filter must fail.


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 selects only among peers that are **currently connected**. The way you write the peers determines how the DRA chooses between them.

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 does not change. You must define each peer in the DRA's Diameter peer configuration. Name each peer by its **exact** hostname, which is case-sensitive. Each peer must be currently connected. The DRA skips disconnected peers.

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

- `:failover` - always the first connected peer in the list. This gives a deterministic primary, then a secondary.
- `:random` - a random connected peer. This spreads load evenly.

Priority and weight

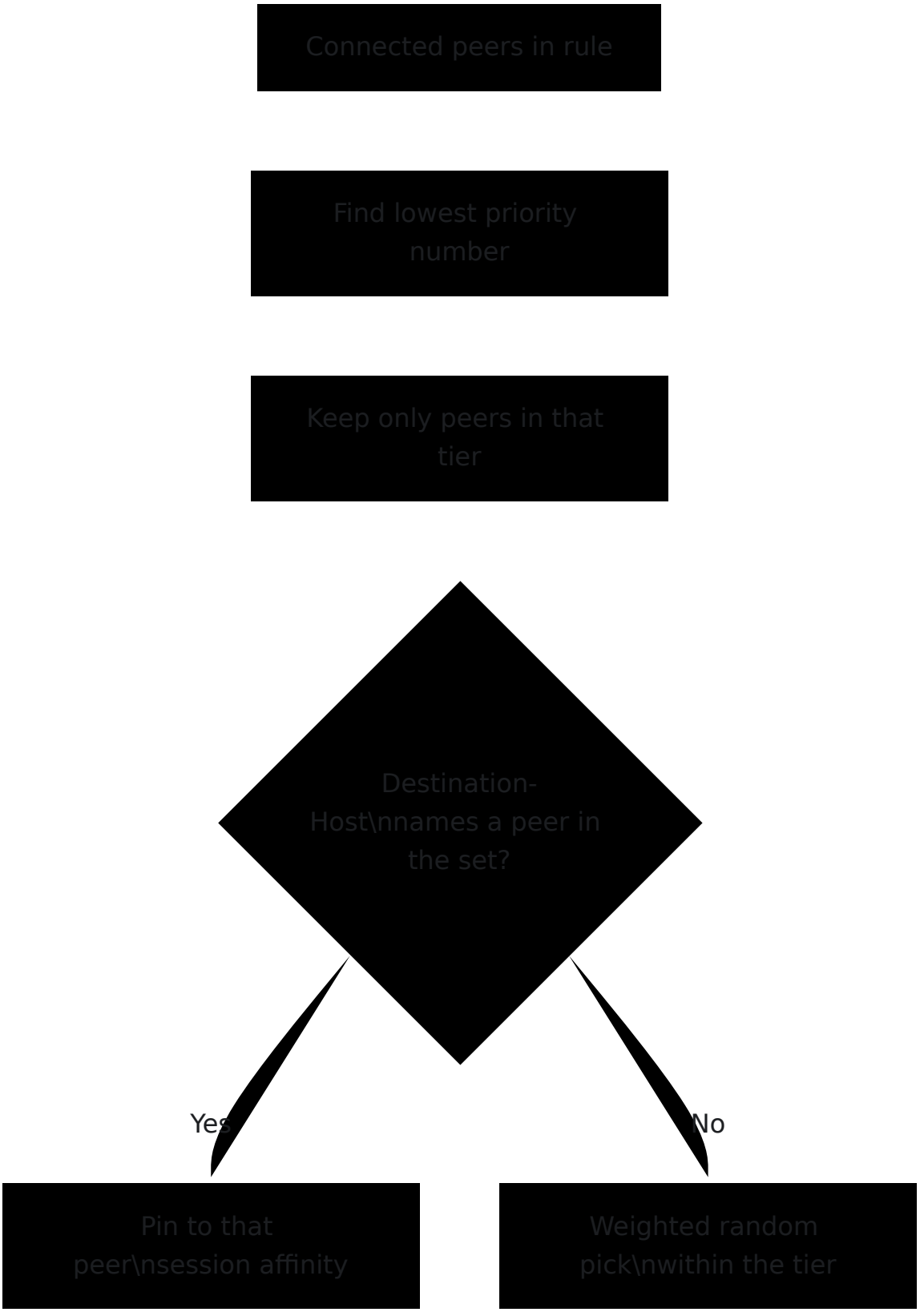
For per-rule failover and load balancing, write the peers as maps with `priority` and `weight`. This is independent of the global algorithm.

```
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. The DRA considers only connected peers.
2. The **lowest priority number** present wins the tier. A lower number is preferred, as with DNS SRV records. Higher-numbered tiers act as backups. The DRA uses a backup tier only after every connected peer in a lower tier is gone. This provides **failover**.
3. Within the winning tier, the DRA chooses one peer at random, **weighted by `weight`**. Equal weights spread load evenly. Unequal weights bias the split. This provides **load balancing**.

If you omit `priority` or `weight`, each one defaults to `100`.



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

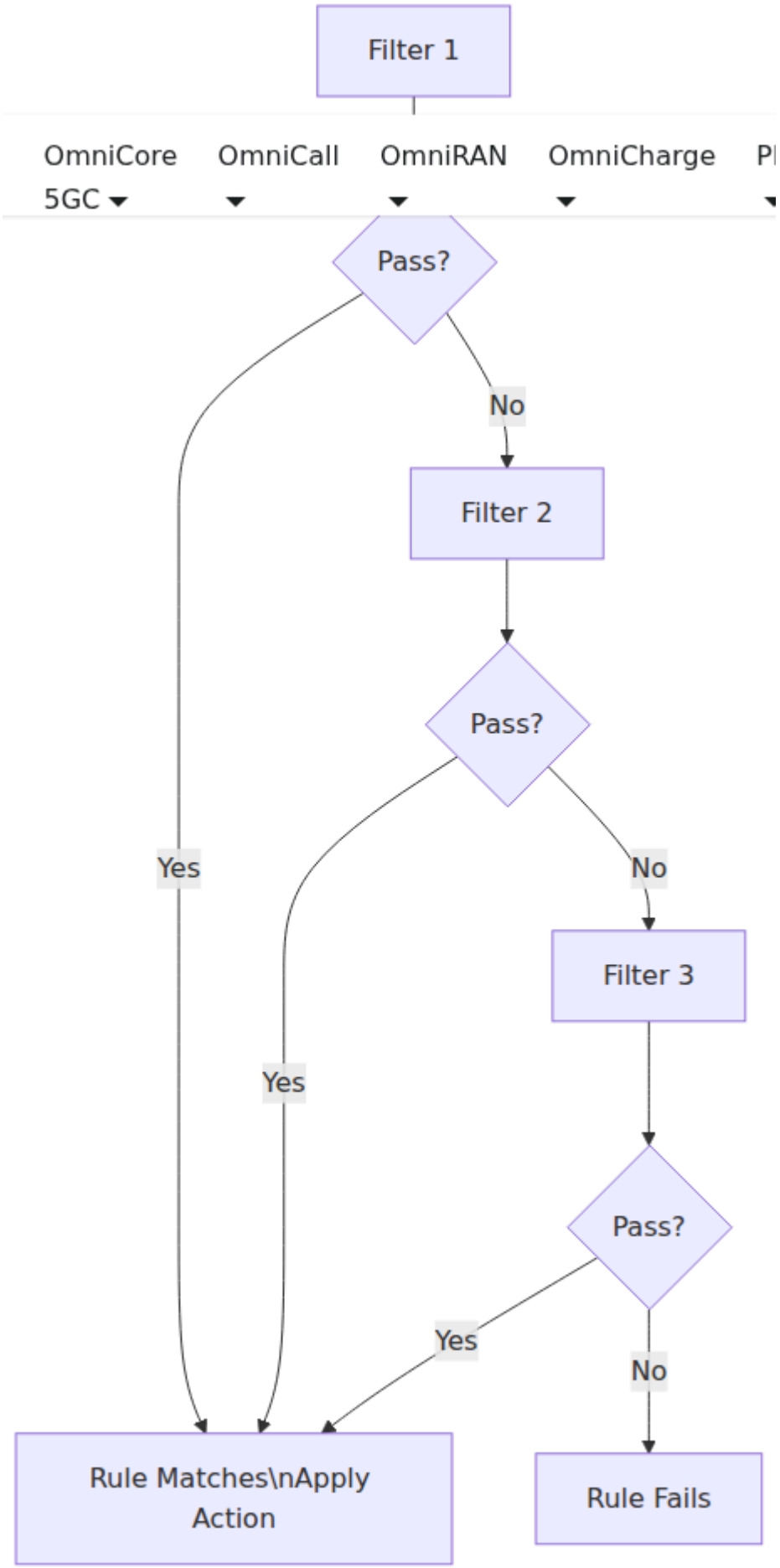
Session Affinity (Gx / Gy)

Stateful applications carry one **Session-Id** for the life of a session. Mid-session messages must reach the **same** server that handled the session-initiating request. If the DRA re-balances a mid-session message to a different peer, that peer rejects it with **DIAMETER_UNKNOWN_SESSION_ID (5002)**.

Advanced Routing preserves affinity with 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 DRA **pins** the request to that peer. This overrides priority and weight.
- If there is no such Destination-Host, the DRA load balances the request normally.

The session-initiating request (for example Gx CCR-Initial) carries no Destination-Host. The DRA load balances it across the tier. The answer returns the chosen server's Origin-Host. The client then places that value in Destination-Host on all later requests (CCR-Update, CCR-Terminate). This pins those requests to that server automatically. In Gx/Gy, the **CC-Request-Type AVP (416)** distinguishes the request type.


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. This behaviour does not affect them.

Configuration

Configure Advanced Routing 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`, the DRA evaluates no rules and all traffic uses standard routing.
`rules`	List	Yes	`[]`	Ordered list of routing rules. The DRA evaluates them 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. The DRA shows it in logs and the management UI.
`match`	Atom	Yes	-	How the filters combine: `:all` (logical AND), `:any` (logical OR), or `:none` (logical NOR). See Match Modes.
`filters`	List	Yes	-	The conditions that a message must meet. See Filters.
`route`	Map or Atom	Yes	-	Where the DRA sends matching traffic. This is a peer list or a special destination. See Route Destinations.

Filters

A filter is a tuple of `{filter_type, value}`. Advanced Routing supports these filters:

Filter	Example	Description
`:via_peer`	`{:via_peer, "mme01.example.com"}`	The Origin-Host of the peer that sent the request (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 can be a list. The DRA treats the list as a logical OR over its members. For example, `{:command_code, [317, 318]}` matches command code 317 or 318. `{:via_peer, ["dra01.example.com", "dra02.example.com"]}` matches either peer.
- **Regex values.** AVP filter values and list members support regular expressions through a regex sigil, for example `{:avp, {1, ~r"999001.*"}}`. You can mix members, for example `{:avp, {1, ["505571234567", ~r"999001.*", ~r"505057.*"]}}`.
- **Presence check (`:any`).** A value of `:any` matches when the attribute is present with any value. For example, `{:avp, {264, :any}}` matches whenever the Origin-Host AVP exists. `{:via_peer, :any}` matches any originating peer.

AVP filter limitations

Use AVP filters for **simple** AVPs only, such as User-Name, Origin-Host, and Destination-Realm. **Grouped AVPs are not supported** and do not match. 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 can 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 the DRA uses standard routing.
Drop	`:drop`	Discard the message with no response to the requesting peer. Use this to blackhole unwanted traffic, such as test subscribers or unwanted ranges.
Error	`{:error, 3004}`	Respond with a Diameter error answer that carries the given Result-Code. The DRA populates Origin-Host, Origin-Realm and Session-Id automatically. The DRA does not forward the message 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 the meanings.

Peer Parameters (priority/weight form)

Parameter	Type	Required	Default	Description
`host`	String	Yes	-	The peer's Origin-Host. It must match the peer's configured Diameter identity exactly and is case-sensitive.
`priority`	Integer	No	`100`	The tier. The DRA prefers lower numbers. It uses higher-numbered tiers only when every connected peer in a lower tier is unavailable.
`weight`	Integer	No	`100`	The share of traffic within a tier. The DRA splits traffic 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. Fall 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, the DRA answers requests 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: The DRA splits new sessions (CCR-Initial) roughly 50/50 between the two PCRFs. The DRA pins mid-session messages to the PCRF that

handled the session. It uses the Destination-Host that the messages carry.

Weighted Load Balancing with a Backup

Send more traffic to a larger PCRF and less to a smaller one. 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, the DRA splits new sessions 70/30 between them. Traffic moves to `pcrf-dr` only when **both** priority-10 PCRFs are down. Established sessions stay pinned to their server for their lifetime.

Route by Originating Peer

Route S6a traffic that arrives from specific upstream DRAs to a local HSS pair. This is 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: The DRA routes to the local HSS nodes only S6a requests that arrived from `dra01` or `dra02` and carry the matching Origin-Realm (AVP 296). 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: The rule matches S6a messages from the given Origin-Realm whose User-Name is exactly `505571234567` or begins with `999001`. The DRA routes these messages 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 DRA sends the request to the peer named in its Destination-Host AVP (293). If the Destination-Host AVP is missing, the DRA treats the match as a failure and applies standard routing.

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: The DRA drops S6a messages whose IMSI begins with `999999`. It sends no response. Use this to filter test traffic or to block 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: The DRA answers S6a traffic from the named peer with **DIAMETER_TOO_BUSY (3004)**. This signals the peer to back off. The DRA forwards nothing upstream.

Block a Specific Command

Reject a particular command type with an explanatory error instead of a silent drop.

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

How it works: The DRA answers S6a Purge-UE-Requests (command 321) with **DIAMETER_UNABLE_TO_COMPLY (5012)**. This disables that operation and leaves other commands untouched.

Reading the Active Configuration

You can read the active rules 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 behaves:

- `route.selection` is `"priority_weight"` when the rule uses priority/weight maps. Otherwise it is the global algorithm (`"failover"` or `"random"`).
- In `priority_weight` mode, the API returns each peer with its effective `priority` and `weight`. It fills in the defaults 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. The first match wins.

2. **Common matches early.** Place the highest-volume matches near the top. This reduces evaluation overhead.
3. **Validate regex.** Test regular-expression patterns against captured messages before deployment.
4. **Name clearly.** Use descriptive `rule_name` values. Logs and the management UI then explain themselves.
5. **Prefer priority/weight for stateful apps.** Use priority/weight peer maps for Gx/Gy. The DRA then applies session affinity automatically.

Troubleshooting

Sessions Rejected with UNKNOWN_SESSION_ID

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

Possible causes:

- The clients (PCEF/OCF) do not set Destination-Host on mid-session requests, so the DRA cannot pin them.
- A rule routes this traffic with bare hostnames and the global `:random` algorithm. This algorithm does not apply session affinity.

Resolution:

1. Confirm that mid-session requests carry a Destination-Host that names the server that answered the initial request.
2. Convert the rule's peer list to priority/weight maps. The DRA then applies session affinity automatically.

Matched Traffic Answered with 3002

Symptoms: A rule should route certain requests, but the DRA answers them 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 match the peers' Origin-Host values exactly.

Resolution:

1. Check the peer connection state in the management UI or logs.
2. Confirm that the `host` values in the rule match the peers' Origin-Host exactly. The match is case-sensitive.

A Rule Is Not Taking Effect

Symptoms: Traffic that should match a rule goes through 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. This can be a case mismatch on an AVP value, or an unsupported grouped AVP.

Resolution:

1. Confirm that `enabled` is `true`. Restart the DRA after the change.
2. Review the rule order. The first matching rule wins.
3. Confirm the filter values against a captured message. Remember that the DRA cannot match grouped AVPs.

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. For example, a mid-session message reached the wrong server	RFC 6733 §7.1.5
5012	DIAMETER_UNABLE_TO_COMPLY	The DRA cannot honour the request.	RFC 6733 §7.1.5

Code	Name	Description	Reference
		Use this to reject specific commands	

Relevant AVPs

AVP	Code	Description	Reference
Destination-Host	293	Names the specific peer that must receive a request. This is 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

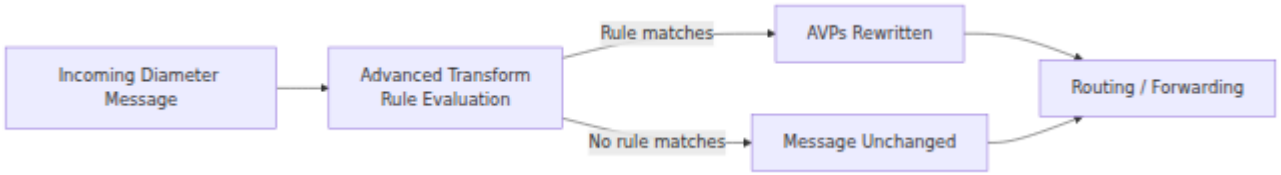
The **Advanced Transform** module rewrites Attribute-Value Pairs (**AVPs**) inside Diameter messages that match operator-defined criteria. Use it to correct, translate, or normalise message contents as they pass through OmniDRA. You do not need to change the sending or receiving network elements.

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, the DRA applies its transform to the message and stops rule processing.

Advanced Transform provides three transform actions. `:edit` changes the value of an existing AVP. `:remove` deletes one or more AVPs by code. `:overwrite` rewrites one or more AVPs by name, including grouped AVPs. See Transform Actions.

Advanced Transform shares its rule-processing model with the routing engine. See Advanced Routing for how the DRA makes routing decisions.

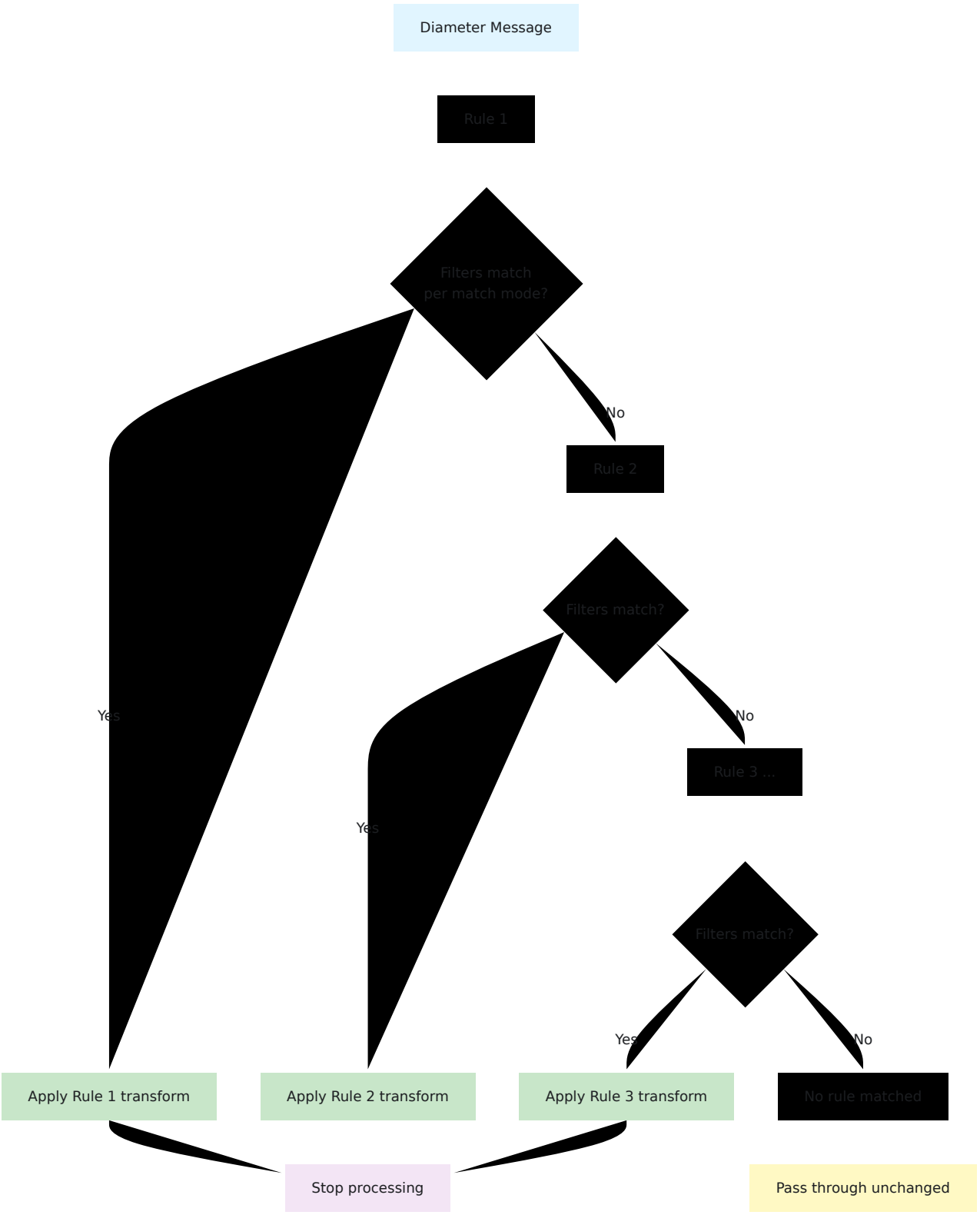


How Rules Are Processed

The DRA evaluates rules **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, the DRA applies its transform and considers no further rules. 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:

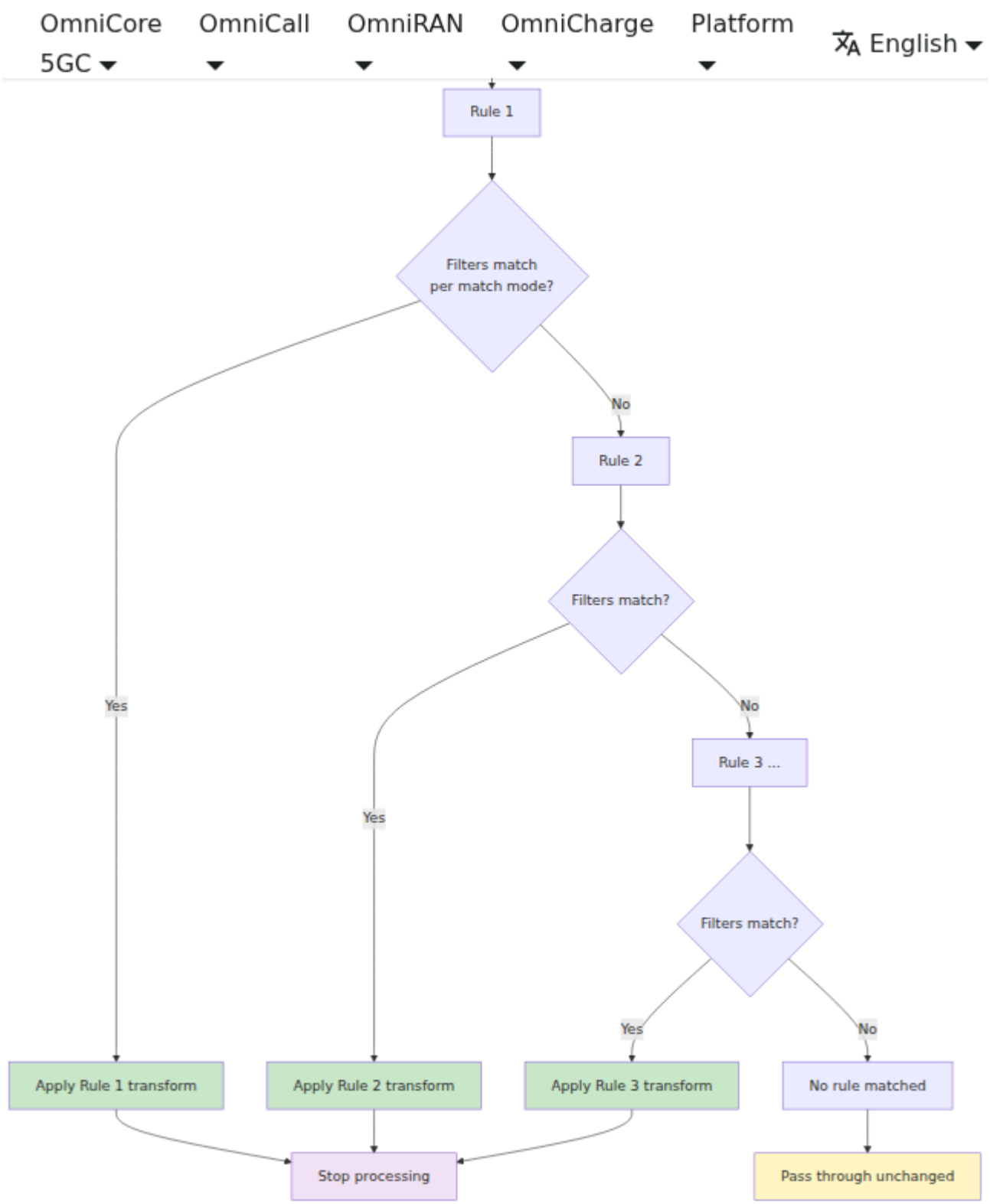
- `:all` - logical **AND**. Every filter must pass for the rule to match.
- `:any` - logical **OR**. At least one filter must pass for the rule to match.
- `:none` - logical **NOR**. No filter may pass for the rule to match (inverse matching).



Transform rules operate on the message contents. They do not select a destination. A rule can rewrite an AVP, for example a realm or identity. The routing engine then uses that AVP 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. Advanced Transform provides six filter types.

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. It matches **request** packets only and never matches answers. It accepts a list of hostnames (OR). See Peer Filtering.
`:from_peer`	`{:from_peer, "hss01.example.com"}`	Matches the **peer that answered** a message. It matches **answer** packets only and never matches

Filter	Format	Description
		requests. It 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 has the structure `{avp_id, avp_value}`. The `avp_id` is the numeric AVP code. The `avp_value` is the value to compare against, expressed as a string.

Use the `:avp` filter for **simple AVPs** such as `User-Name` and `Origin-Host`. **Grouped AVPs are not supported** and do not match. Complex binary values do not match.

The filter matches values 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. This is 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**, not the message contents. They are direction-specific and complementary:

- **`:to_peer`** matches the **resolved destination peer** that the DRA is about to forward a request to. It matches **request** packets only. It never matches an answer.
- **`:from_peer`** matches the **peer that answered** a message. It matches **answer** packets only. It never matches a request.

Both filters accept a single hostname or a **list of hostnames**. With a list, the filter matches if the peer equals **any** hostname in the list. This is 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` map describes the transform. Its `action` key selects **one of three** transformations. The **direction** of the message determines which actions are permitted:

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 affects only AVPs that **already exist** in the message. If the targeted AVP is not present, the DRA makes **no change** and passes the message on as received.
- Each entry in `avps` is `{:avp, {<code>, <new_value>}}`. It 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.
- It matches by **AVP code only**. It **ignores** the value in each `avps` entry 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. It uses the supplied `dictionary` to decode and re-encode the message.
- It decodes the full message, so `:overwrite` can reach **grouped / record AVPs** that `:edit` cannot.
- It **requires** a `dictionary` key. This key names 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`, the DRA uses only `<code>` and ignores the value (use `:any`).
`dictionary`	Atom	Only for `:overwrite`	`:not_used`	The Diameter dictionary that decodes and re-encodes the message so the DRA can overwrite named (grouped) AVPs, for example `:diameter_gen_3gpp_s6a` The `:edit` and `:remove` actions ignore this key.

Configuration

Configure Advanced Transform under `module_advanced_transform` in `config/runtime.exs`. The module is disabled by default. It does nothing until you set `enabled` 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`, the DRA evaluates no rules and all messages pass through unchanged.
`rules`	List	Yes (when enabled)	-	Ordered list of transform rules. The DRA evaluates them 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. It supports operational clarity and monitoring.
`match`	Atom	Yes	-	How the filters combine: `:all` (AND), `:any` (OR), or `:none` (NOR). See How Rules Are Processed.
`filters`	List	Yes	-	List of filter conditions that a message must satisfy 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`. On a match, the `:edit` action rewrites `User-Name` to `999999000000002`. If a message carries no `User-Name` AVP, the DRA makes no change.

Use case: Correct or normalise a specific subscriber identity as it transits the DRA. For example, map 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`). The routing engine then forwards the message toward the correct partner network. The first matching rule wins, so place more specific IMSI ranges above broader ones.

Use case: Direct roaming or MVNO subscriber traffic to the correct hosted core network. The DRA selects the network by the 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**. The `:from_peer` filter limits it to answers from `hss01.partner.example.com`. The `:command_code` filter 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: Remove an AVP from a partner network's answers. Use this when the partner populates the AVP incorrectly, or when the AVP must not go downstream.

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 the message. It decodes the full message, so `:overwrite` can reach AVPs nested inside grouped/record structures that `:edit` cannot touch.

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

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	The realm that the message is destined for. The DRA uses it 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. This reduces 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. This confirms that the 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)**. It sits at the centre of a Diameter signalling network. It relays request and answer messages between clients such as MME/SGSN nodes and back-end services such as the HSS, PCRF, and OCS. OmniDRA is built in Elixir/Erlang on the OTP `:diameter` library. It operates as a **Diameter Relay** and forwards messages between peers. It does not terminate the applications that the messages carry.

Table of Contents

What the DRA Is

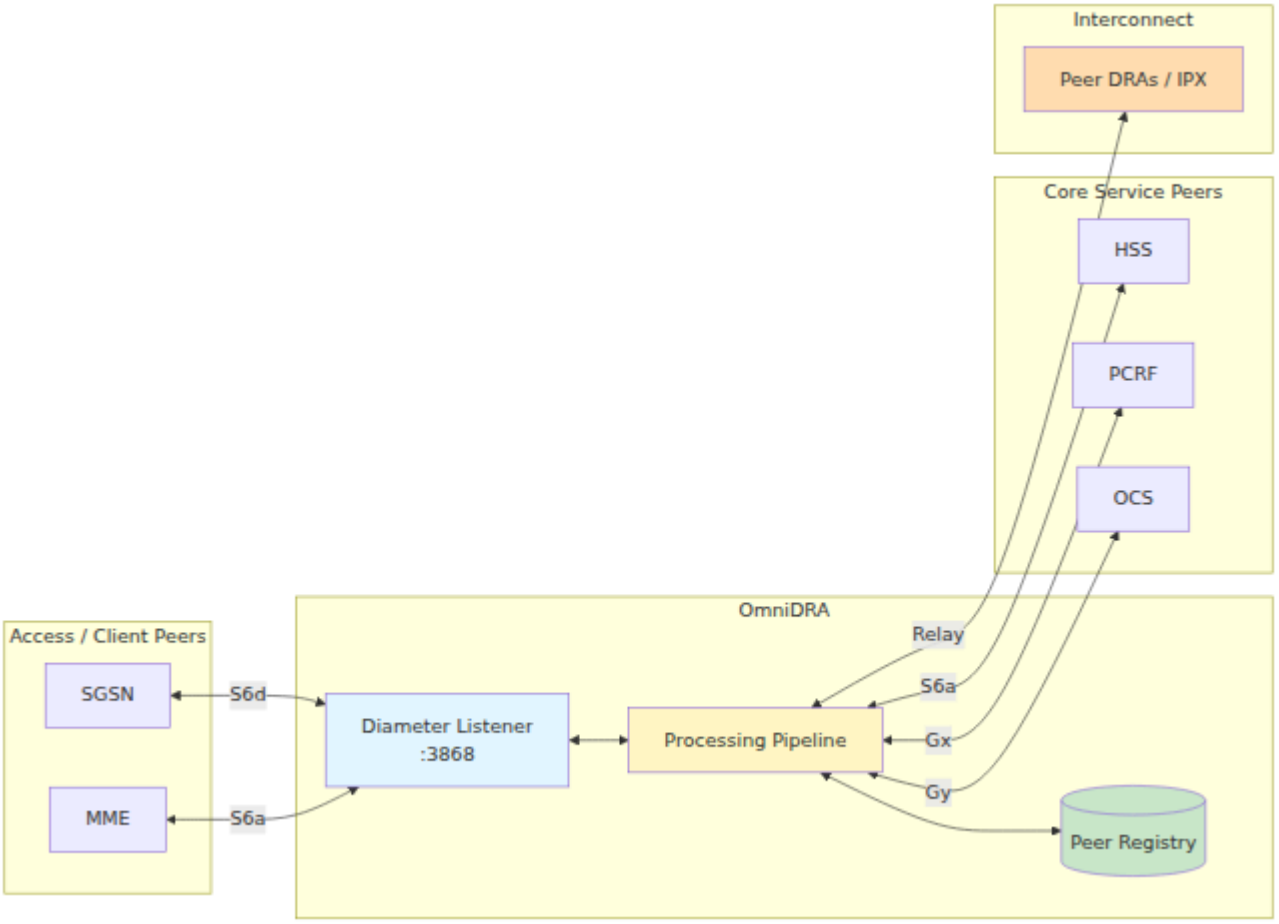
A **Diameter Relay** receives Diameter messages on a listening socket. It decides which connected peer must handle each message. It then forwards the message unchanged, aside from the routing AVPs that the relay manages. OmniDRA runs as a relay, so it does not implement the application logic of the interfaces it carries. It routes **S6a**, **Gx**, **Gy**, and any other Diameter application transparently. The peers must advertise support for the application.

Key characteristics:

- **Standards-based routing** - Request routing follows the priority mechanism of the Diameter Base Protocol (RFC 6733). It selects a peer with the **Destination-Host**, **Destination-Realm**, and **Application-Id**.
- **Stateful answer correlation** - Answers always return along the reverse path of their request. The DRA correlates each answer by the **Hop-by-Hop Identifier**.
- **Optional processing modules** - Roaming steering, subscriber lookup, advanced routing, and transform modules extend the standard behaviour. They do not replace it.
- **Security pipeline** - Rate limiting, Diameter firewalling, AVP sanitisation, and topology hiding protect the network at the signalling edge.

Network Context

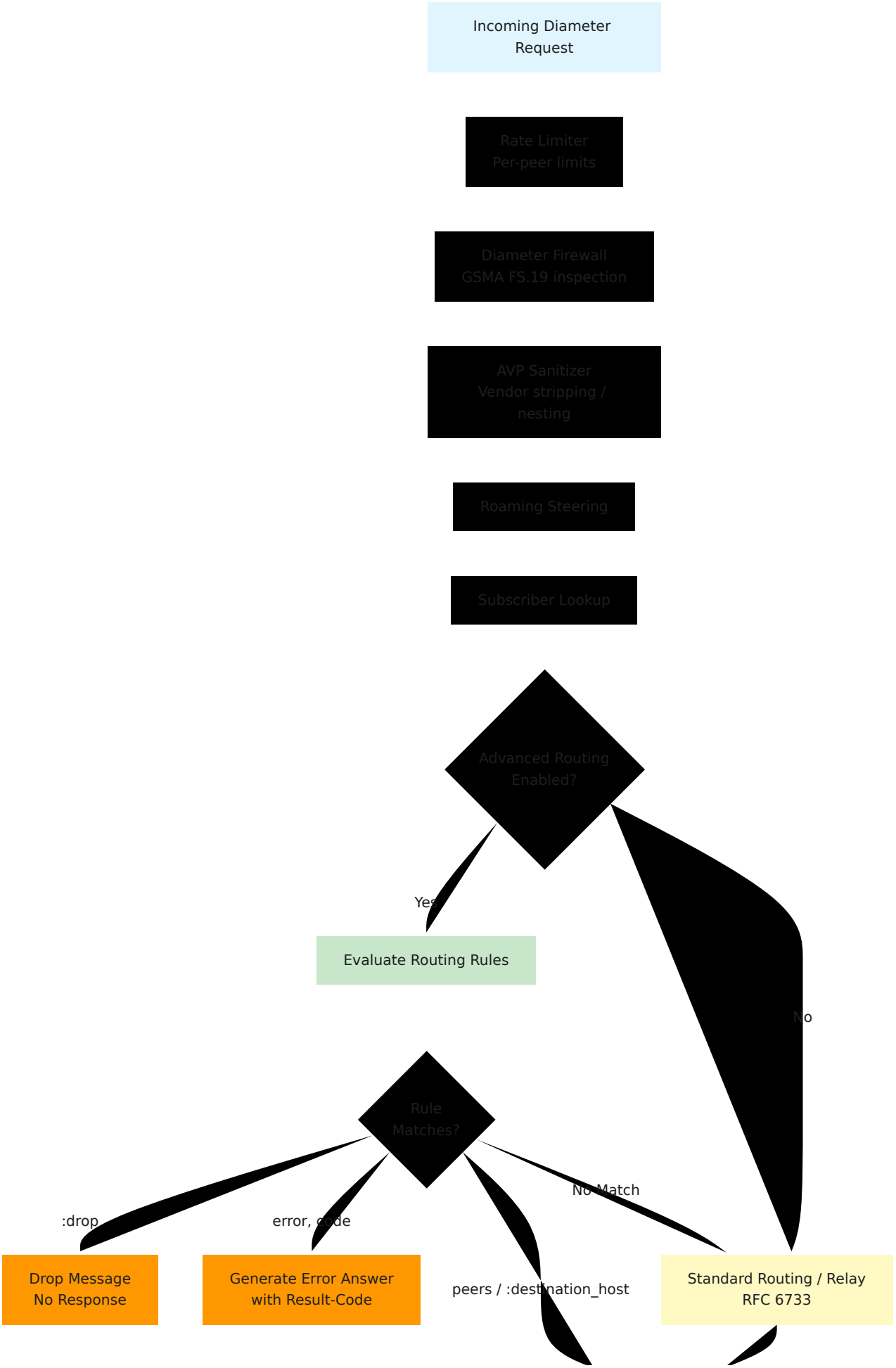
Deploy the DRA between the **access-side** signalling nodes (such as MME and SGSN) and the **core-side** services (such as HSS, PCRF, and OCS). Every peer establishes a Diameter connection to the DRA. The DRA never originates application traffic of its own.



The DRA listens for incoming Diameter connections on TCP/SCTP port `3868`. This is the standard port defined by RFC 6733. Each peer advertises the realms and applications it supports during **Capabilities Exchange** (CER/CEA). The DRA records this in its peer registry for use in routing decisions.

Request-Processing Pipeline

Each incoming request passes through an ordered pipeline. The security and steering stages run first. The routing stages run next. Answers bypass this pipeline. 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 before 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 that holds its session. This is the only stage that runs *after* Advanced Routing. 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 DRA discards the request with no response. An `error` outcome generates a Diameter answer that carries the configured **Result-Code**. Otherwise, the DRA forwards matched requests to the specified peers or the **Destination-Host** peer. Unmatched requests fall through to standard routing.

Rx Session Binding is the one exception to Advanced Routing having the last word. Its scope is narrow. It only redirects a request that was already going to be relayed, and only for Rx. It leaves a `:drop` outcome, an `error` outcome, and an explicit `Destination-Host` 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 the request contains this AVP, 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 the Destination-Host is absent, the DRA selects a connected peer that advertises the target realm. The DRA load-balances across the matching peers.
3. **Application-Id** - The DRA filters the candidate peers to those that advertise support for the message's application. The peers advertise this support 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**. The registry tracks every peer connection and the capabilities that the peer advertised during Capabilities Exchange. The registry is the authoritative source for routing decisions. Only peers in the **connected** state are eligible to receive traffic. The DRA automatically excludes disconnected or down peers.

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**.

The DRA learns realm and application support dynamically from CER/CEA. To add capacity for an interface, connect an additional peer that advertises the relevant realm and application. The DRA starts to load-balance to the new peer automatically.

Message Relaying

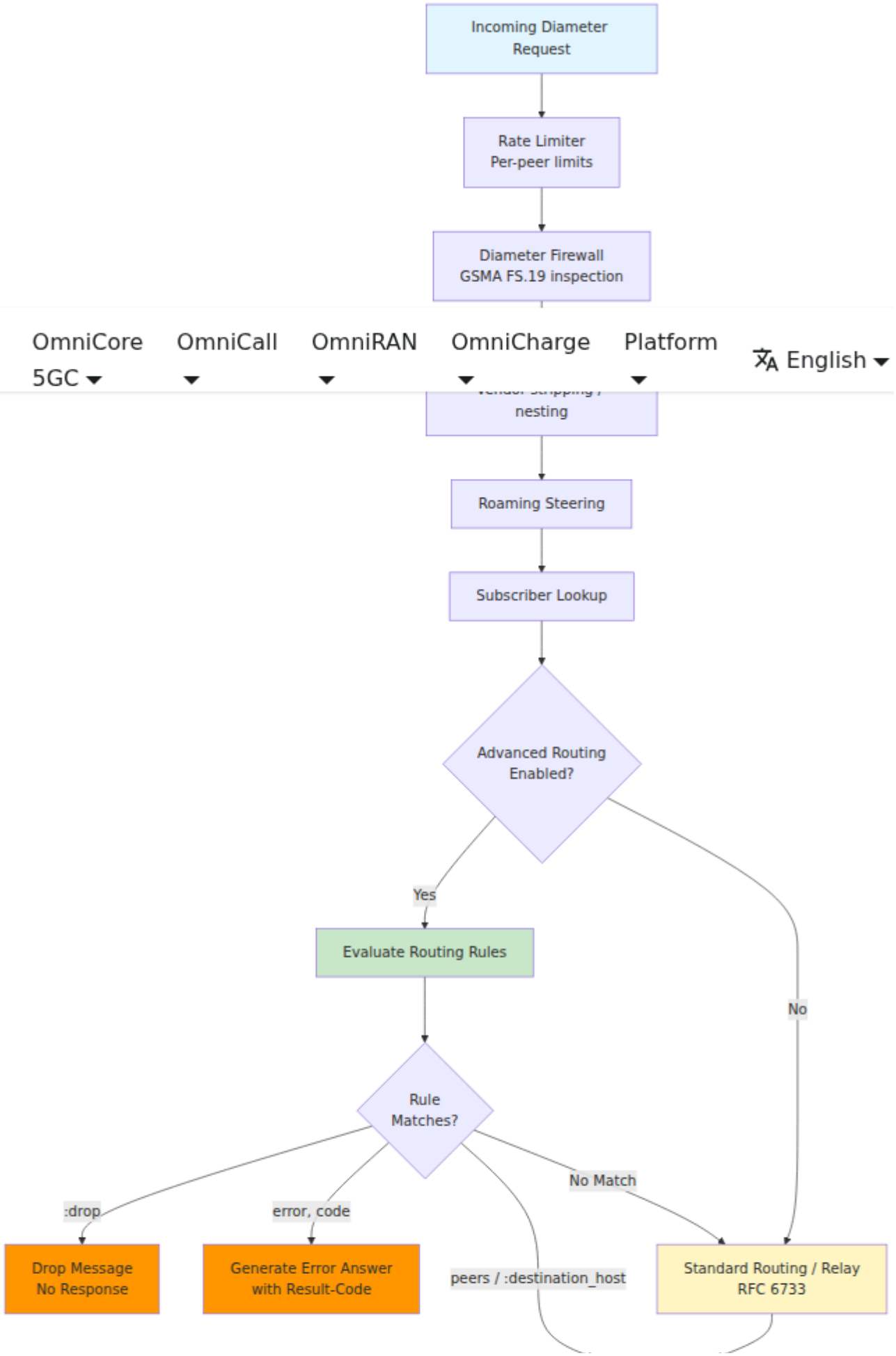
The DRA relays two fundamentally different classes of message, and treats them differently.

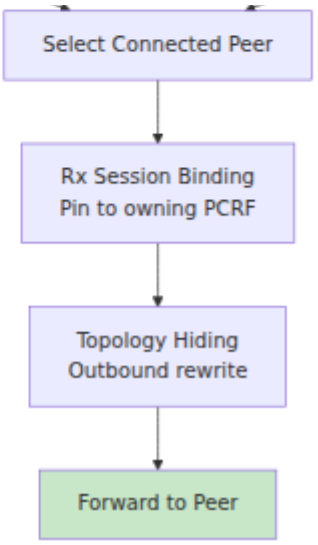
Requests

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

Answers

Answers never re-enter the routing pipeline. Instead, the DRA correlates each answer to its originating request. The DRA then returns the answer 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** stays unchanged across every hop.
- **Hop-by-Hop routing** - The **Hop-by-Hop Identifier** changes at each hop. The DRA uses it to maintain routing state and to 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. The DRA cleans up this state after it relays the answer.

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.

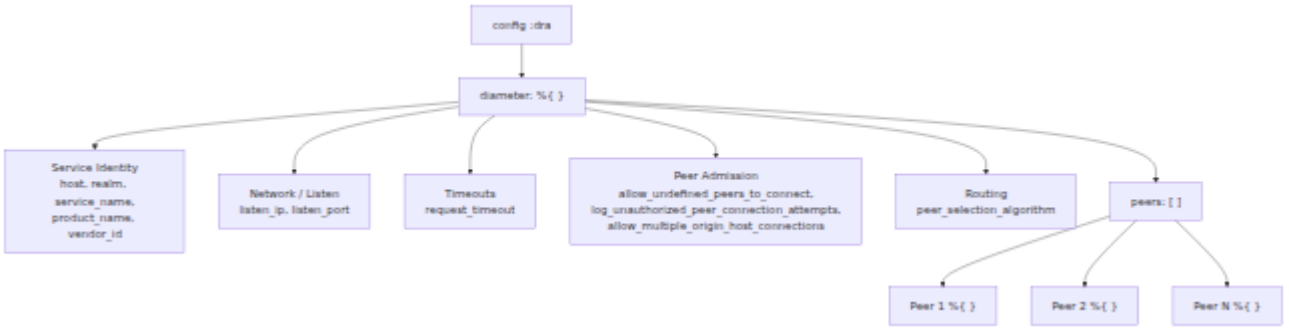
- **Advanced Routing** - Rule-based routing, drop/error outcomes, and peer targeting.
- **Configuration** - Base DRA configuration, transports, and module settings.
- **Security** - Rate limiting, GSMA FS.19 Diameter firewall, AVP sanitisation, and topology hiding.

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 that the agent makes. Declare it 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. It also contains a `peers` list. Each entry in the list 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. Each key appears 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`
`peer_selection_algorithm`	Atom	Yes	-

Parameter	Type	Required	Def
`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. Configure it 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 setting affects how the DRA distributes traffic and how it behaves during peer outages. See Standard Routing for the full routing decision flow. See Advanced Routing for rule-based overrides.

Value	Behaviour
`:random`	The DRA chooses a matching peer at random for each request. This spreads load across all eligible peers.
`:failover`	The DRA prefers the first matching peer. It uses later peers only when the higher-priority peers are unavailable.

Peer Configuration

Each entry in the `peers` list is a map that describes a single Diameter peer. The map defines 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	Descriptio
`host`	String	Yes	-	The peer's Diameter Identity (FQD This value mu match exactly and is case-sensitive. Routing and Advanced Routing rules require an exa match.
`realm`	String	Yes	-	The peer's Diameter real
`ip`	String	Yes	-	The peer's primary IP address used the connectio
`additional_ips`	List	No	-	List of IP addresses for multi-homed SCTP association. When unset, t DRA uses `ip`. This key appli only with `transport: :diameter_sc`

Parameter	Type	Required	Default	Descriptio
				See SCTP Multihoming.
`port`	Integer	Yes	-	The peer's Diameter port (typically `386`
`transport`	Atom	Yes	-	Transport protocol for th connection: `:diameter_tc` for TCP, or `:diameter_sc` for SCTP. See the Transport reference bel
`source_ip`	String	No	`listen_ip`	Local IP addre to bind when DRA dials this peer (outbour connections only). Default to the service `listen_ip`.
`source_port`	Integer	No	-	Local port to bind when the DRA dials this peer (outbour connections only). When unset, the OS assigns an ephemeral pc If two peers

Parameter	Type	Required	Default	Descriptio
				share the sam destination IP and port, set explicit value Each connect then uses a distinct local socket.
`initiate_connection`	Boolean	Yes	-	When `true`, t DRA connects out to the pee (client role). When `false`, the DRA waits for the peer t connect (serv role). See Connection Modes.
`origin_host`	String	No	service `host.realm`	The `Origin-Host` the DRA presents **to** th one peer. This key is separat from `host` above, which this peer's ow identity. Set t key when a p expects the D to use a spec host (for example a partner realm

Parameter	Type	Required	Default	Descriptio
				The DRA appl the override t the `CER`/`CEA` and the base- protocol messages on this connectic See Per-Peer Identity.
`origin_realm`	String	No	service `realm`	The `Origin-Realm` the DR presents **to** th one peer. This key is separat from `realm` above, which this peer's ow realm. Set thi key with `origin_host` on its own. Se Per-Peer Ident

> **SCTP multihoming:** a peer that uses `:diameter_sctp` can bind multiple local and remote addresses for resilience. SCTP Multihoming describes the multihoming settings and per-peer address lists.

Transports

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

Connection Modes

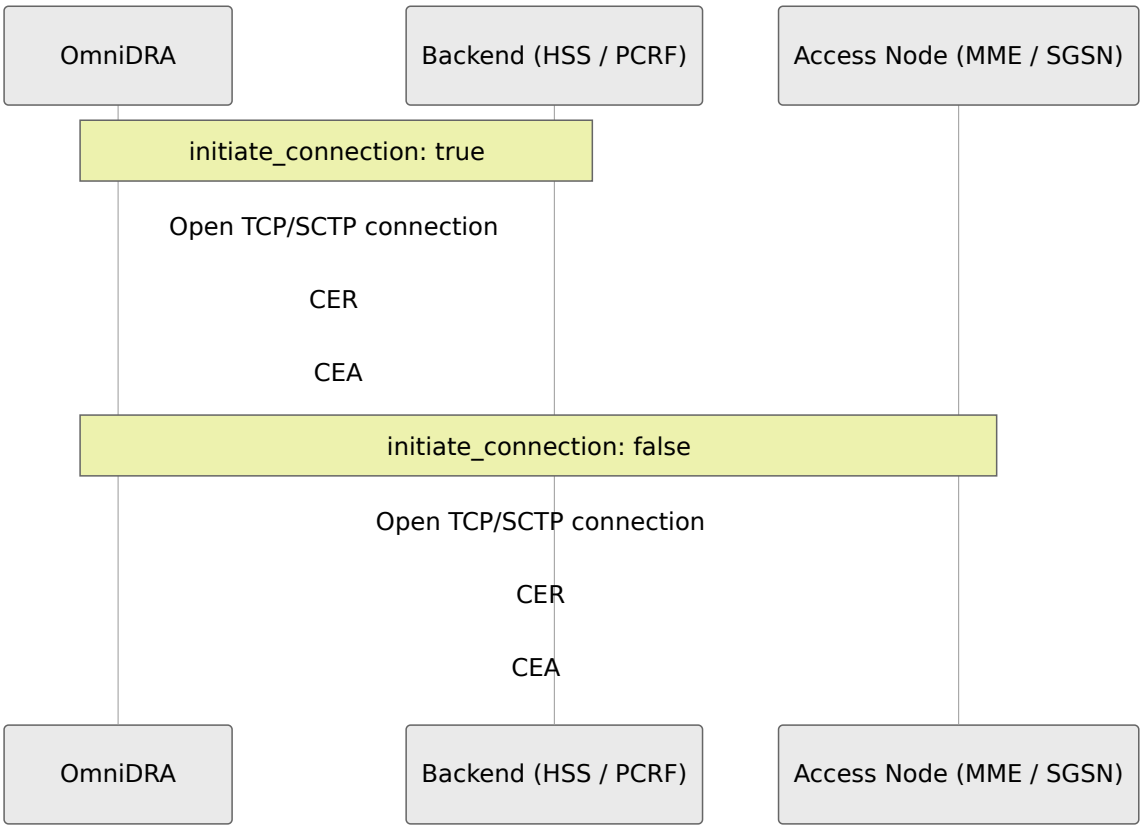
The `initiate_connection` key controls whether the DRA dials a peer or waits to be dialled.

Initiate Connection (`initiate_connection: true`)

- The DRA acts as the Diameter client and initiates the TCP/SCTP connection to the peer.
- Use this mode for backend systems such as an HSS or PCRF.
- If the peer is unreachable, the DRA retries the connection.

Accept Connection (`initiate_connection: false`)

- The DRA acts as the Diameter server and waits for the peer to connect.
- Use this mode for access-side nodes such as an MME, SGSN, or P-GW.
- The peer must be present in `peers`, unless you set `allow_undefined_peers_to_connect` to `true`.
- The DRA differentiates inbound peers by the `Origin-Host` that they advertise in the CER. Two Diameter nodes can share one source IP. The DRA matches them independently when each presents a distinct `host`.



Per-Peer Identity

The DRA has one service-wide identity. It combines `host` and `realm` into the `Origin-Host` (for example `dra01.production.omnitouch`) and advertises `realm` as the `Origin-Realm`. By default the DRA presents this identity to every peer.

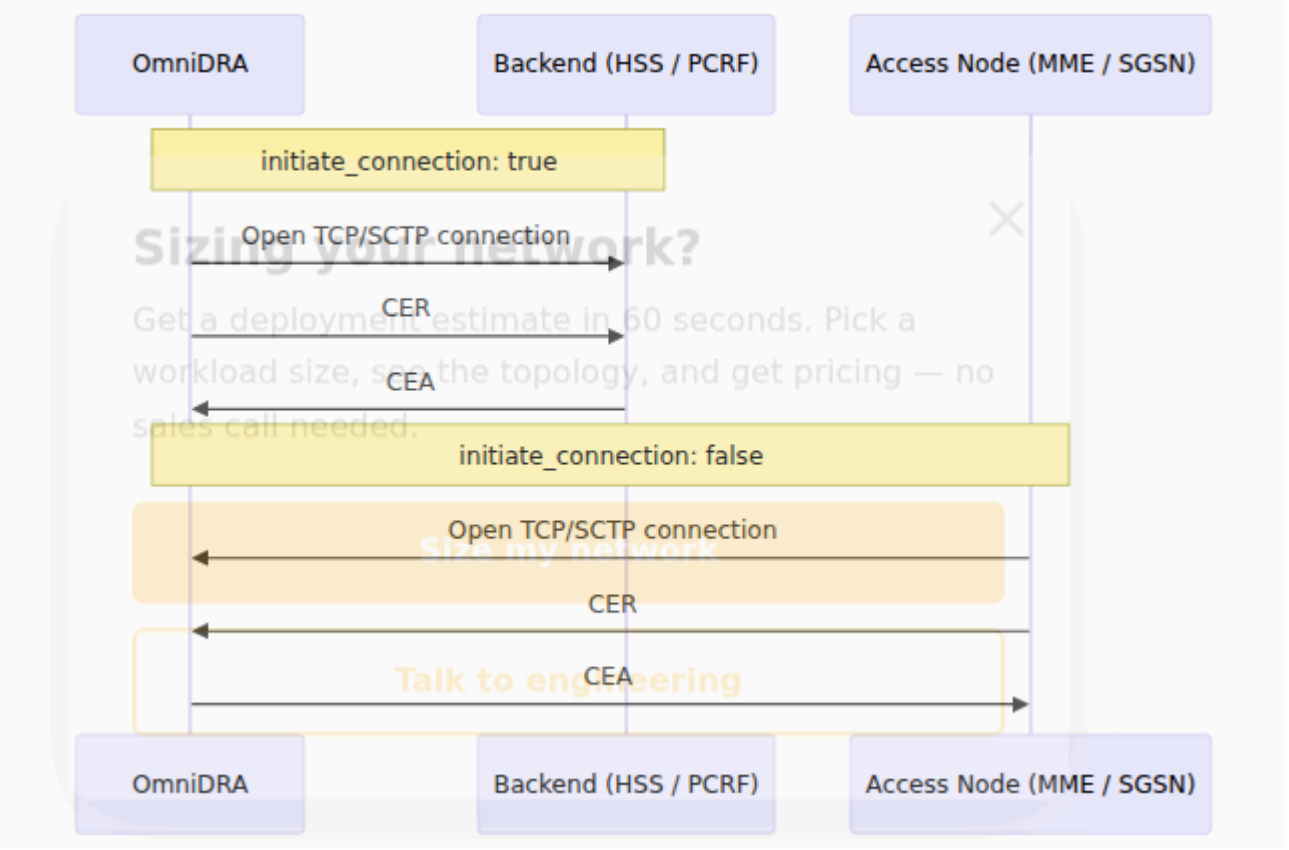
Some peers expect the DRA to present a different identity. An interconnect or roaming partner often requires the DRA to use a host and realm from the partner naming scheme. Set `origin_host` and `origin_realm` on that one peer to change the identity the DRA presents on that connection.

These two keys are the identity the DRA presents **to** the peer. They are separate from the peer's own `host` and `realm`, which are the identity the peer advertises **to** the DRA. The DRA applies the override to that one connection only. Every other peer continues to see the service-wide identity.

```
peers: [
  %{
    # This peer's own identity (what it advertises to the DRA):
    host: "hss.partner.example",
    realm: "partner.example",
    ip: "10.0.0.9",
    port: 3868,
    transport: :diameter_sctp,
    initiate_connection: true,
    # The identity the DRA presents TO this peer:
    origin_host: "dra-partner.epc.mnc001.mcc001.3gppnetwork.org",
    origin_realm: "epc.mnc001.mcc001.3gppnetwork.org"
  }
]
```

See Per-Peer Parameters for the `origin_host` and `origin_realm` parameter definitions.

The diagram below shows the DRA presenting its default identity to one peer and an override identity to a partner peer.



Behaviour

The override changes the identity in the Capabilities Exchange (`CER` / `CEA`) on that connection. It also changes the identity in the base-protocol messages on that connection: the watchdog messages (`DWR` / `DWA`) and the disconnect messages (`DPR` / `DPA`).

You can set either key on its own. An unset key falls back to the service value. If you set only `origin_realm`, the DRA changes only the realm and keeps the service-wide `Origin-Host`.

The DRA uses the override `Origin-Host` for loop detection on that connection. If a request already carries a `Route-Record` AVP that matches the override, the DRA still detects the loop per RFC 6733 §6.1.8.

The DRA fixes the identity when it sends the `CER`. If you change an override and reload the configuration at runtime, the DRA re-establishes that one connection so the new identity takes effect. Connections to other peers are not affected.

Complete Example

The following example configures a DRA. The DRA accepts connections from an MME over SCTP. It 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** - The DRA matches peer `host` values exactly and case-sensitively. Rules in Advanced Routing must reference the same `host` string that you configure here.
- **Capabilities exchange** - On connection, the DRA and each peer exchange supported applications and identities through CER/CEA. The DRA advertises the `host`, `realm`, `product_name`, and `vendor_id` values at this stage.
- **Vendor-ID 10415** - This is the standard 3GPP `Vendor-Id`. It is the expected value for 3GPP Diameter interfaces.
- **Peer selection** - When multiple peers match a request, `peer_selection_algorithm` decides which one the DRA uses. 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

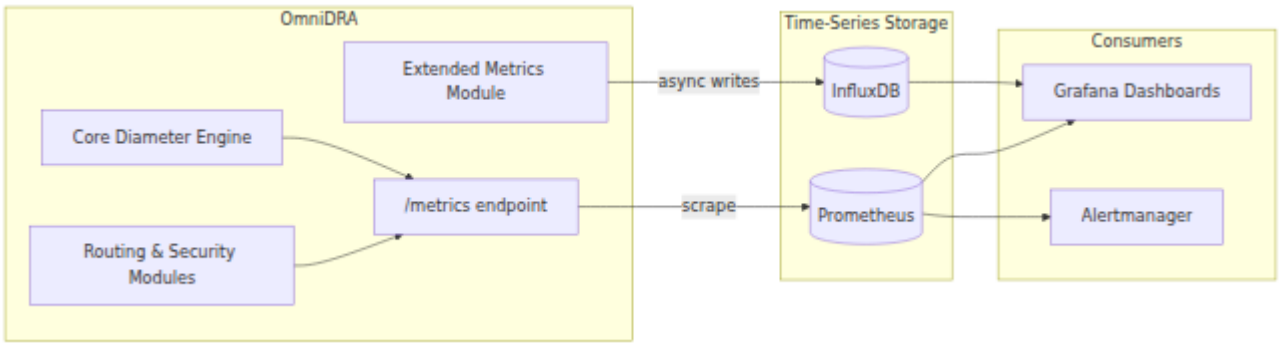
- SCTP Multihoming - Multihomed SCTP transport and per-peer address configuration.
- Standard Routing - How the DRA selects peers and routes requests, including `peer_selection_algorithm` behaviour.
- Advanced Routing - Rule-based routing overrides keyed on peer `host` values.

Metrics & Monitoring

OmniDRA exposes comprehensive **Prometheus** metrics. They cover 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. Use them to build dashboards and alerts.

Observability Flow

OmniDRA publishes all metrics in Prometheus exposition format at the `/metrics` endpoint. Prometheus scrapes this endpoint on a regular interval and stores the samples. Dashboards or alerting tools then query the resulting time series.



Prometheus scrapes the **standard Prometheus metrics** from `/metrics`. The **Extended Metrics Module** publishes high-cardinality, subscriber-level telemetry directly to **InfluxDB**. This keeps the Prometheus series count manageable (see Extended Metrics Module).

> OmniDRA also exposes **BEAM/Erlang runtime metrics** on `/metrics`. This page does not document them. All counters are cumulative. Query them with `rate()` to get per-second rates.

Core Diameter Metrics

These metrics are always available. They 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 that 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 that sent the message.
`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 that holds the most recent delay per peer. It is 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 that the DRA sent but that the peer did not answer within the timeout period.

Labels:

Label	Description
`origin_host`	Original requester.
`routed_to`	Peer that did not 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 that advanced routing rules produce. 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 that the advanced routing `:drop` action dropped. The DRA sends no response.

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 the DRA drops traffic:

- An advanced routing rule matches with a **drop** action.
- The DRA discards the message.
- The DRA sends no answer message to the requesting peer.

Rate Limiting & Overload

Error Responses

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

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 the DRA generates error responses:

- 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 DRA sends the answer back to the requesting peer. It does not forward the answer.

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. Use it to analyse Diameter traffic patterns beyond the standard metrics. It currently tracks LTE subscriber attach attempts. It correlates S6a Authentication Information Request (AIR) and Answer (AIA) message pairs.

These metrics carry **high-cardinality** fields such as IMSI. The module therefore writes them asynchronously to **InfluxDB** and does not expose them on the Prometheus `/metrics` endpoint. This keeps the Prometheus series count bounded and avoids metric explosion.

Configuration

The module is disabled by default. Configure it 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` DR by the
`attach_attempt_reporting_enabled`	Boolean	No	`false`	En tra rep LTI att (S(AI

InfluxDB Connection

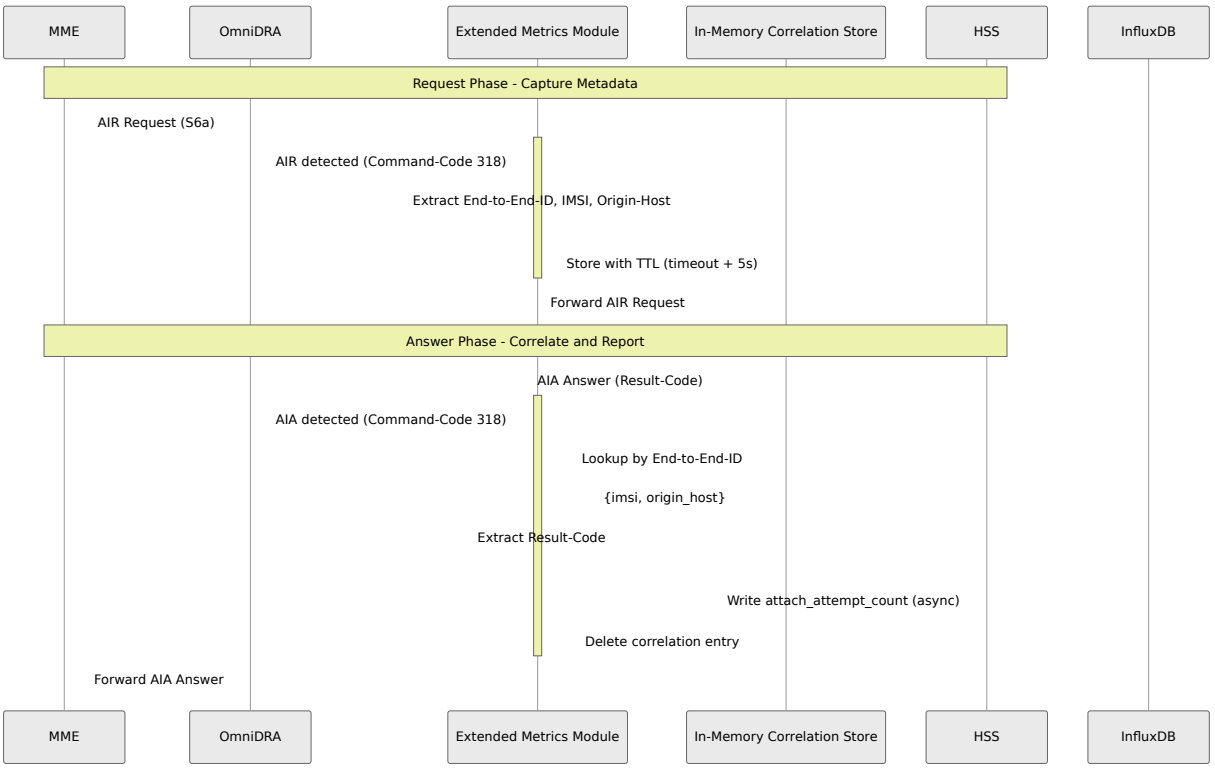
The Extended Metrics Module writes asynchronously to InfluxDB through an Instream connection (Instream). Configure the target under `config :dra, DRA.Metrics.InfluxDB`.

```
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 tha line-protocol InfluxDB. Us InfluxDB's U port (`8089` b not the HTT (`8086`).
`writer`	Module	No	`Instream.Writer.Line`	Instream wr module. `Instream.Wr` writes with t protocol.

Attach Attempt Tracking

The module monitors S6a AIR/AIA pairs. It correlates each request with its answer through the Diameter End-to-End-ID. It then emits a subscriber-level metric. The metric carries 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 that have attach failures.
- **Network Performance Monitoring** - Monitor attach attempt patterns by origin (MME/SGSN).
- **Roaming Analytics** - Analyse inbound roaming attach success rates.

Operational Notes

- Attach attempt metrics track only S6a AIR/AIA pairs (Application-Id `16777251`, Command-Code `318`).
- Request metadata expires after the configured request timeout plus 5 seconds.
- Metric processing is asynchronous. It does not block message flow.
- The module operates independently from the 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 through InfluxDB.
- OmniDRA also exposes **BEAM/Erlang runtime metrics**. This page does not document them.
- All counters are cumulative. Query them with `rate()` for per-second rates.

- Only the Extended Metrics Module uses high-cardinality tags such as IMSI. This avoids 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

This module controls how quickly a Diameter peer becomes routable after it connects. It also prevents traffic to peers that flap. A flapping peer connects and disconnects repeatedly.

Background

Before OmniDRA routes to a peer, the peer 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. The peer stays there until it passes successive watchdog (DWR/DWA) exchanges. The watchdog timer **Tw** paces these exchanges, roughly 30 seconds with the standard 30000 ms default. This hold applies even though CER/CEA already succeeded.

OmniDRA removes that implicit delay. The watchdog promotes reconnecting peers immediately. OmniDRA replaces the delay with a **settle time**. The settle time is an explicit, configurable hold that applies after the peer connects. During the hold, the peer is connected but not routable. If it disconnects within the window, OmniDRA cancels the hold and sends no traffic to it.

Configuration

Both keys are optional and live in the `:diameter` config map. If you set neither key, 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 the DRA routes traffic to it. `0` routes immediately. A positive value holds the peer for that duration. It cancels the hold if the peer 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. This costs more watchdog traffic.

Set the settle time no higher than the worst-case flap interval that you must ride out. A longer value increases flap protection. It 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). It reports down until it becomes routable. The metric therefore reflects what the DRA can route.

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 it settled. The DRA routed no traffic.
`destination peer is settling; request not yet routable...`	A request arrived for a peer that is still settling. The DRA answered `DIAMETER_UNABLE_TO_DELIVER (3002)`.
`no suitable peer to route request...`	The DRA cannot route the request to a destination that is **not** settling (down, unknown, or misconfigured).

The last two lines 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)

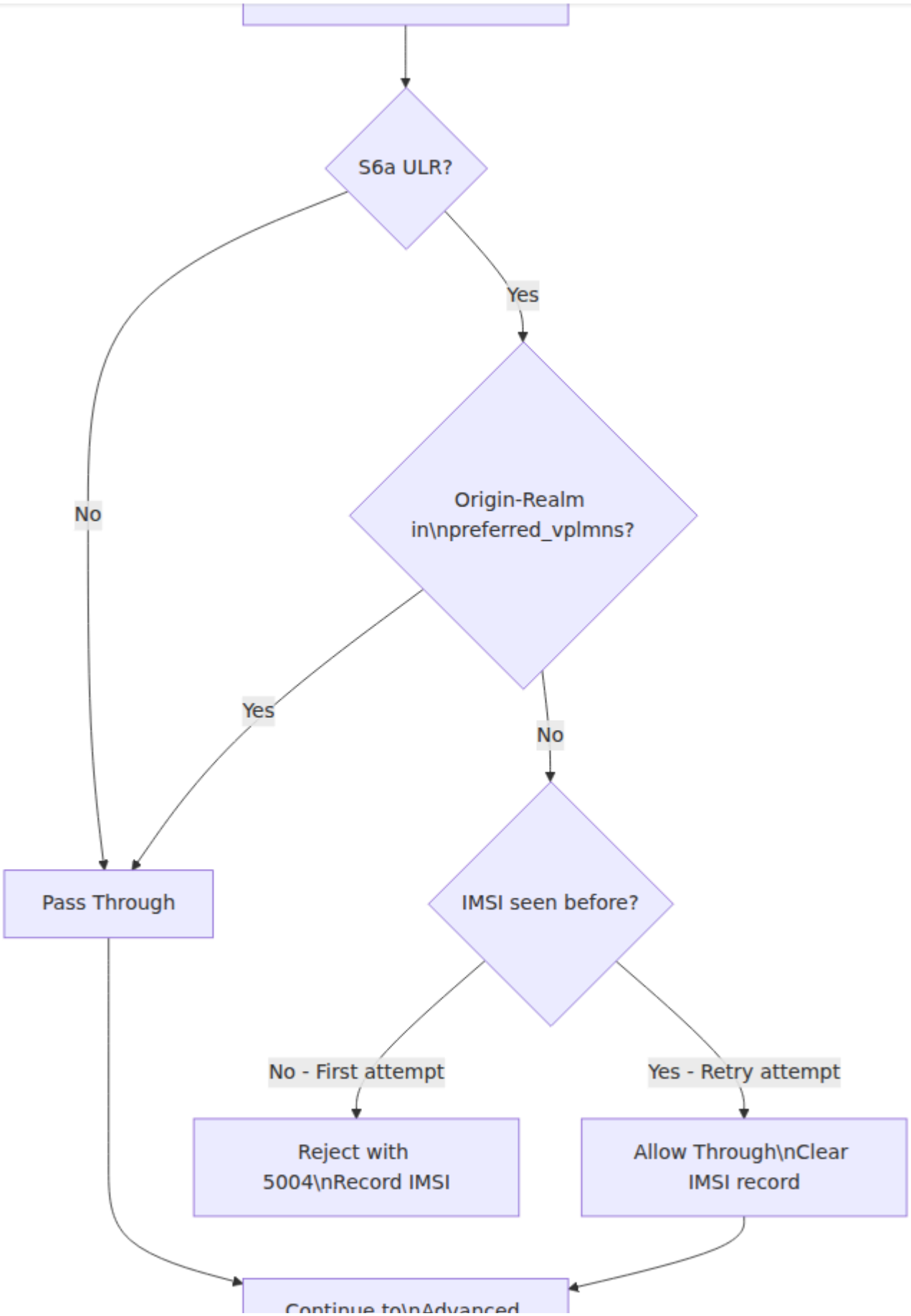
Steering of Roaming (SoR) lets the HPLMN influence which VPLMN a roaming subscriber attaches to. When a subscriber tries to register through a non-preferred VPLMN, the DRA intercepts the S6a Update-Location-Request (ULR). It rejects the ULR with **DIAMETER_ERROR_ROAMING_NOT_ALLOWED (5004)**. This causes the UE to detach and re-attach. The UE then selects a preferred VPLMN if one is available.

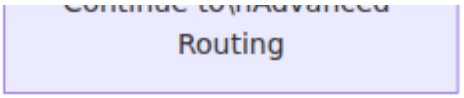
If preferred coverage is unavailable and the subscriber returns to the same non-preferred VPLMN, the module allows the request through on the next attempt.

3GPP TS 29.272 Section 5.2.1.1 and 3GPP TS 23.122 define SoR.

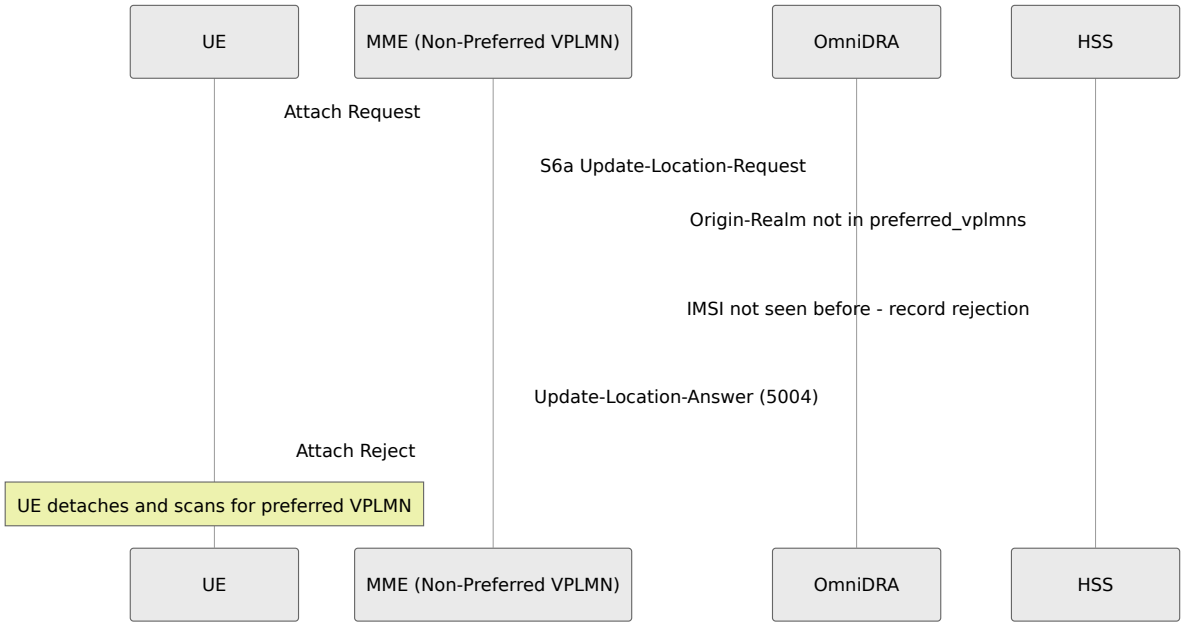
How It Works

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

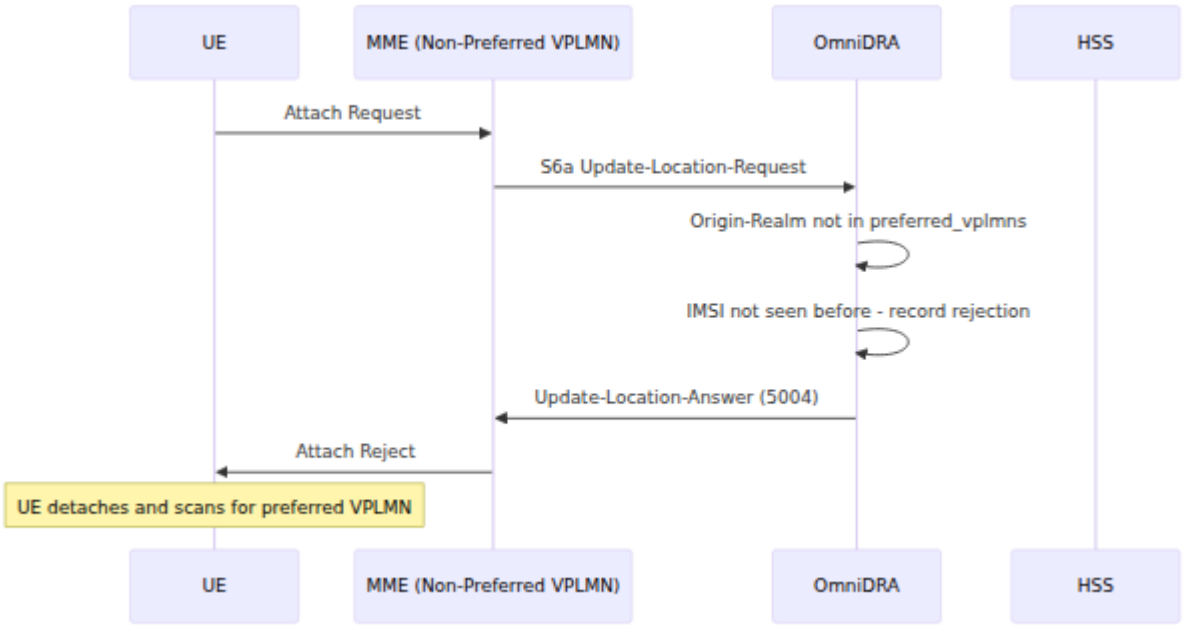

S6a ULR?
Yes
No
Origin-Realm in\npreferred_vplmns?
Yes
No
Pass Through
IMSI seen before?
No - First attempt
Yes - Retry attempt
Reject with 5004\nRecord IMSI
Allow Through\nClear IMSI record
Continue to\nAdvanced



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, that message never reaches Advanced Routing.

Subscriber Tracking

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

When a tracked IMSI reaches the `max_rejections` threshold and sends another ULR, the module allows the request through. It then deletes the tracking record.

If no further ULR arrives within the TTL window, the record expires. The next cleanup cycle removes it. The module then treats a later ULR from the same IMSI as a first attempt and rejects it again.

Configuration

Configure the module 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 that when enabled. `[DRA.Router.M` the IMSI-trackir
`max_rejections`	Integer	No	`1`	Number of time before the moc the default of `1` ULR and allows
`rejection_ttl_seconds`	Integer	No	`300`	Time in second expires. Cleanu subscriber does the module tre attempt. This v cleanup interva
`rejection_result_code`	Integer	No	`5004`	Diameter Resu answer. `5004` c DIAMETER_ERR per 3GPP TS 29
`preferred_vplmns`	List	No	`[]`	List of Origin-R preferred VPLM allows a ULR fr Origin-Realm th

Configuration Examples

Reject Once, Then Allow

This is the default behaviour. The module rejects a subscriber on a non-preferred VPLMN once. If the subscriber returns, the module forwards the ULR 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 arrives from an MME in `epc.mnc003.mcc310.3gppnetwork.org`, which is not in the preferred list. The module rejects it with Result-Code 5004. The UE detaches and tries to find a preferred network. If it returns to the same non-preferred VPLMN within 300 seconds, the module forwards the ULR normally.

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

Multiple Rejections Before Allow

Use this when the operator wants the UE to make multiple attempts before it falls 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 module rejects the subscriber three times. It then allows the subscriber through on the fourth attempt. The TTL extends to 600 seconds for the additional retry cycles.

Use case: Dense urban areas where preferred coverage exists. The UE may need multiple attach attempts to acquire it.

Metrics

Rejections

Metric: `diameter.roaming_steering.reject.count` **Type**: Counter **Description**: The DRA increments this counter each time the SoR module rejects a ULR. **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**: The DRA increments this counter when the module allows a ULR through. **Labels**:

- `origin_realm` - Origin-Realm of the originating VPLMN

- `reason` - Why the module allowed the request: `preferred_vplmn` or `max_rejections_reached`
- `imsi` - IMSI of the subscriber. Present only when the reason is `max_rejections_reached`

Error Responses

Metric: `diameter.roaming_steering.error.count` **Type**: Counter **Description**: The DRA increments this counter when the processor returns an error answer message for 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: The module repeatedly rejects subscribers on non-preferred VPLMNs. They never attach successfully.

Possible causes:

- `rejection_ttl_seconds` is too short. The tracking record expires before the subscriber retries.
- `max_rejections` is higher than the number of retries that the UE performs before it gives up.

Resolution:

1. Increase `rejection_ttl_seconds` to suit the UE's retry timing.

2. Confirm that `max_rejections` is a value that the UE can reach within its retry cycle.

Subscribers on Preferred VPLMNs Being Rejected

Symptoms: The module rejects ULRs from preferred partner networks with 5004.

Possible causes:

- The Origin-Realm of the preferred VPLMN is not in `preferred_vplmns`.
- The Origin-Realm string does not match exactly. The match is case-sensitive.

Resolution:

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

Module Not Taking Effect

Symptoms: The module forwards ULRs from non-preferred VPLMNs without rejection.

Possible causes:

- `enabled` is set to `false`.
- The module process is not running. Check the supervisor.

Resolution:

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

Reference

Diameter Codes

Code	Name	Description	Refer
5004	DIAMETER_ERROR_ROAMING_NOT_ALLOWED	The subscriber may not roam in this VPLMN	3GPP 29.27 Sectio 7.4.4

S6a Commands

Command-Code	Name	Description	Reference
316	Update-Location-Request/Answer (ULR/ULA)	The MME sends this 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`. It defines **no** `Subscription-Id` and **no** `Framed-IP-Address` in the message at all. This second point is structural. 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. Some AF implementations do not send one. A representative capture shows these AVPs:

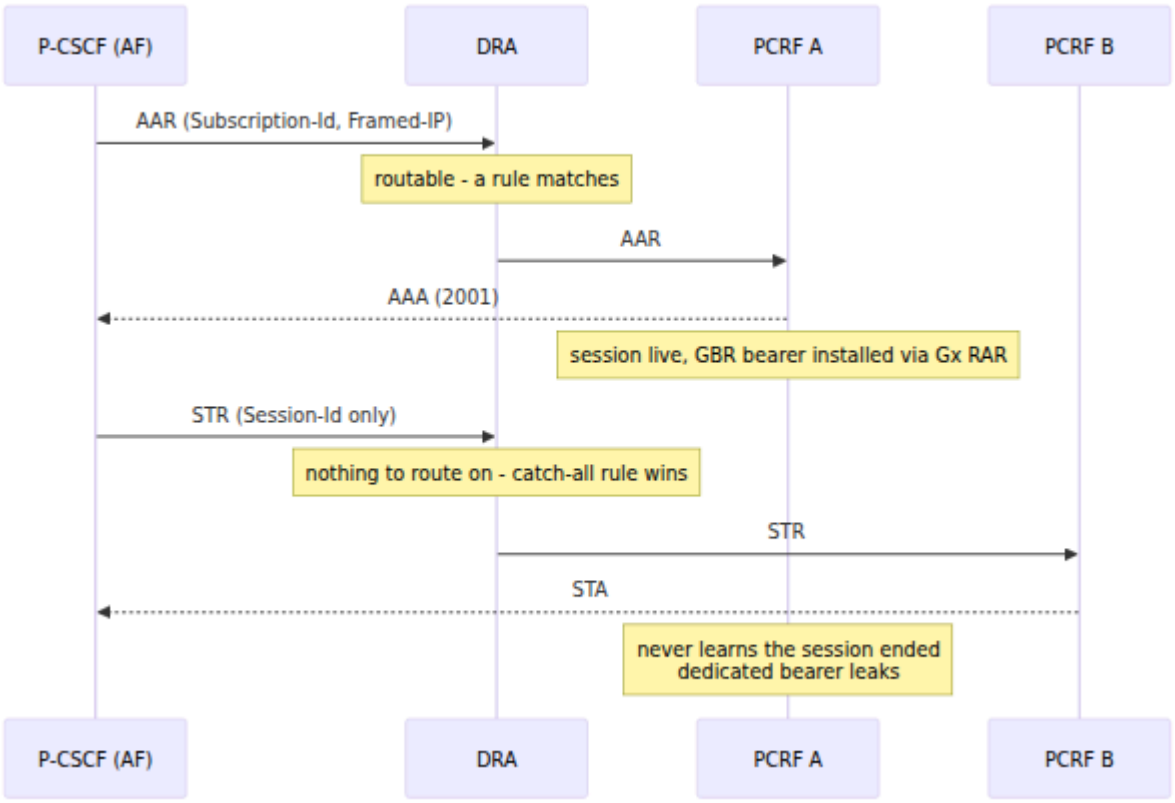
`Session-Id`, `Origin-Host`, `Origin-Realm`, `Destination-Realm`, `Auth-Application-Id`, `Vendor-Specific-Application-Id`, `AF-Application-Identifier`, `Termination-Cause`.

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. It identifies neither the subscriber nor the server.

Compare this with the `AA-Request` that opened the session. That request carries `Subscription-Id` and `Framed-IP-Address`, so an Advanced Routing rule can route it. Compare it also with **Gx**. The `CC-Request` ABNF (TS 29.212 §5.6.2) includes `[ Destination-Host ]`, `*[ Subscription-Id ]` **and** `[ Framed-IP-Address ]`. This 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". A bypass is only something that "may be configured".

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



This failure is silent. RFC 6733 §8.4.2 prescribes no result code for an STR that names an unknown session. The §8.1 state machine handles "STR received" from *any* state with "send STA, clean up". A PCRF that answers `2001` for a session it never had is therefore conformant, and real implementations do this. 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 that routes to `peers: [a, b]` selects per message. An AAR and its STR can therefore 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. 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

The DRA binds 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. This has three consequences:

- An AAR answered with an error, or never answered, leaves no binding.
- The DRA releases a session when its STA arrives, not when it routes the STR. RFC 6733 §5.5.4 explicitly sanctions retransmission after a failover. An early release would misroute the retry, the very failure this module prevents.
- A request never rewrites a binding. A transient peer outage therefore cannot silently migrate a live session to a different PCRF.

Release requires **both** conditions. The STA must come from the peer that owns the binding, and its `Result-Code` must be 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 condition 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. The DRA then destroys the binding to the owner while its bearer is still up. This leaves you worse off than with no module, because teardown now *looks* pinned. The ownership condition 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

The module only redirects routing away from a relay decision. Everything else is a deliberate outcome, and the module leaves it 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 the DRA cannot tear down a session while every peer that its rules point at is unreachable. This is the correct trade. An operator who drains a node, or fails closed on purpose, does not ask to be routed around.

Ordering

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

The answer-path observer runs **before** Advanced Transform. Advanced Transform can rewrite or remove `Session-Id` and `Result-Code` on answers. The binding must key on the `Session-Id` that the AF actually used. The module is deliberately insensitive to topology hiding. It reads the answering peer from its Diameter capabilities, not from an `Origin-Host` AVP. An AVP rewrite therefore 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 you may want either one alone.
`ttl_seconds`	Leak guard for sessions whose STA never arrives. It is **not** a session timer.

Size `ttl_seconds` **above** the AF's `Authorization-Lifetime`. The DRA then never unbinds a long call mid-session. A P-CSCF that advertises `Authorization-Lifetime: 36000` needs more than 10 hours. The 12-hour default suits it. Sweeps run every `ttl_seconds / 2` (minimum 60s). The DRA reclaims a leaked binding 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]`	-

A fourth event, `[:diameter, :rx_session_binding, :reap]`, carries a `count`. The DRA emits it only when the TTL sweep finds expired bindings. A non-zero reap count means STAs are going missing. Alert on this count.

The DRA exposes a count of live bindings, and can list 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 rest of the call. The DRA does not share bindings between DRAs. An AF must therefore reach the same DRA for both halves of a session.
- **The DRA binds the adjacent peer.** In relay mode the DRA can append a Route-Record but not insert a `Destination-Host` (RFC 6733 §6.1.9). The DRA therefore cannot pin a PCRF that sits behind another agent. Only that agent can.
- **A late AA-Answer can resurrect a released session.** If the DRA processes an AAR-update's answer after the STA, `insert_new` recreates the row. The session is over, so this is a leaked row, not a misroute. Session-Ids are not reused (RFC 6733 §8.8). The `ttl_seconds` value bounds the row, and it shows up in the `:reap` counter.
- **The DRA cannot release a session while Advanced Routing fails closed** for Rx. The `{:answer_message, 3002}` outcome is indistinguishable from an operator's deliberate `route: {:error, 3002}`. Keep every PCRF that holds live sessions in the rule's peer list. This case then does not arise.

- **`children` is required in config.** `DRA.Supervisor` starts a module's processes only 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.
- **The DRA binds only the AF → PCRF direction.** PCRF-initiated `RAR` and `ASR` carry a mandatory `Destination-Host` (RFC 6733 §8.3.1, §8.5.1) and route on that. They need no binding.
- **The DRA binds only the Rel-7+ Rx application id (16777236).** It does not bind sessions on the legacy Rel-6 Rx/Gq id (16777229).
- **The DRA emits telemetry but does not export it.** The four events above have no `Telemetry.Metrics` definitions, so nothing reaches Prometheus today. This is the same gap that the pre-existing `subscriber_lookup` events have. Use the live-binding count from a remote shell until that is wired up.

SCTP Multihoming

SCTP multihoming provides network redundancy for Diameter transport. It lets each endpoint bind to multiple IP addresses within a single association. If the primary network path fails, SCTP automatically fails over to an alternative path. It does not disrupt the Diameter session.

Why Multihoming Matters

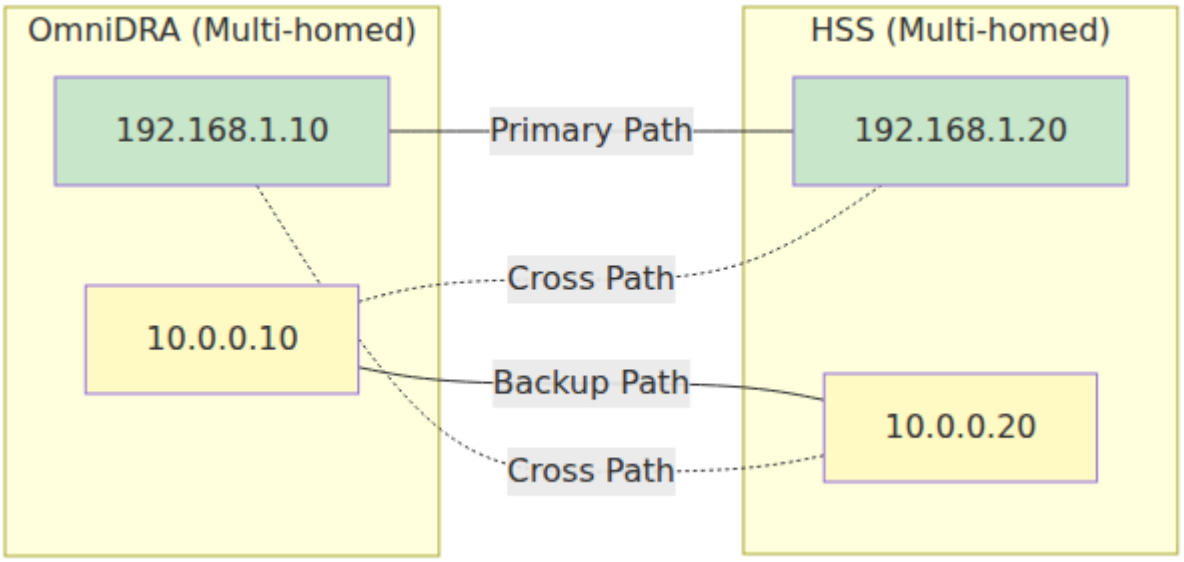
RFC 6733 defines Diameter over SCTP. The multihoming behaviour described here is a native capability of **SCTP** itself. RFC 4960 specifies it. TCP binds an association to a single source/destination IP pair. SCTP lets an association span multiple IP addresses on each side. This gives OmniDRA transport-layer resilience against the loss of an individual network path, NIC, switch, or routed segment. The Diameter connection does not tear down or re-establish.

Key characteristics:

- SCTP **heartbeats** monitor all network paths in the association continuously.
- **Automatic failover** occurs at the kernel level if the primary path becomes unreachable.
- Path switchover does not disrupt the Diameter session. The association and its Diameter state persist.
- The operating system kernel selects the path automatically.

How It Works

OmniDRA and a peer establish a multi-homed association. Each endpoint advertises several IP addresses. The kernel selects a primary path for data transmission. It treats the remaining paths as standby routes. If the primary path fails, the kernel promotes one standby route automatically.



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. They stay available for immediate failover.

Configuration

Configure multihoming in `config/runtime.exs` in two places. The first is the local listen addresses that OmniDRA binds to. The second is 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 a single IP string (backward compatible) or a list of IP strings. Use a list 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**, OmniDRA uses only the first IP in the list. For **SCTP transport**, OmniDRA binds all listed IPs 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. To opt in to multihoming, supply 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 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). It must match the peer's configured identity exactly.
`realm`	String	Yes	-	Peer's Diameter Realm. The DRA uses it 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 for the multi-homed SCTP association. This field is optional. If you omit it, OmniDRA uses only `ip`. For SCTP, include the primary IP in this list. For TCP, OmniDRA

Parameter	Type	Required	Default	Description
				ignores this field, because TCP does not support multihoming.
`port`	Integer	Yes	-	Peer's Diameter port. The 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 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. If you omit `additional_ips`, the peer stays single-homed. Existing peer configurations therefore need no changes.

Complete Example

The following configuration binds OmniDRA to two local addresses. It 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 tables above document `listen_ip`, `listen_port`, and all peer parameters. The remaining top-level parameters control OmniDRA's Diameter identity and peer-selection behaviour. The Configuration Reference documents them:

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 se eli du
`allow_undefined_peers_to_connect`	Boolean	Yes	`false`	Wh on the ma Wh the ac un
`peers`	List	Yes	-	Lis co co Se Co

SCTP Socket Buffers

OmniDRA sets an explicit socket receive and send buffer on every SCTP transport. It uses 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. It is **not** inside the `diameter:` map.

Without an explicit buffer, the SCTP socket inherits a small default `SO_RCVBUF`. This 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`. It

caps the send buffer at `net.core.wmem_max`. To use 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 for both `rec` `:diamete` for TCP. V `net.core` require y

Requirements

SCTP multihoming needs these conditions 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.
- Configure both endpoints for multihoming to achieve full redundancy. A multi-homed OmniDRA connected to a single-homed peer gains path redundancy on the OmniDRA side only.

Limitations

- **TCP does not support multihoming.** With `:diameter_tcp` transport, OmniDRA uses only the primary IP. `listen_ip` uses the first IP in the list, and OmniDRA ignores peer `additional_ips`.
- **Path failover timing** depends on kernel SCTP parameters, such as the heartbeat interval and retransmission thresholds. You tune these outside OmniDRA at the operating-system level.

References

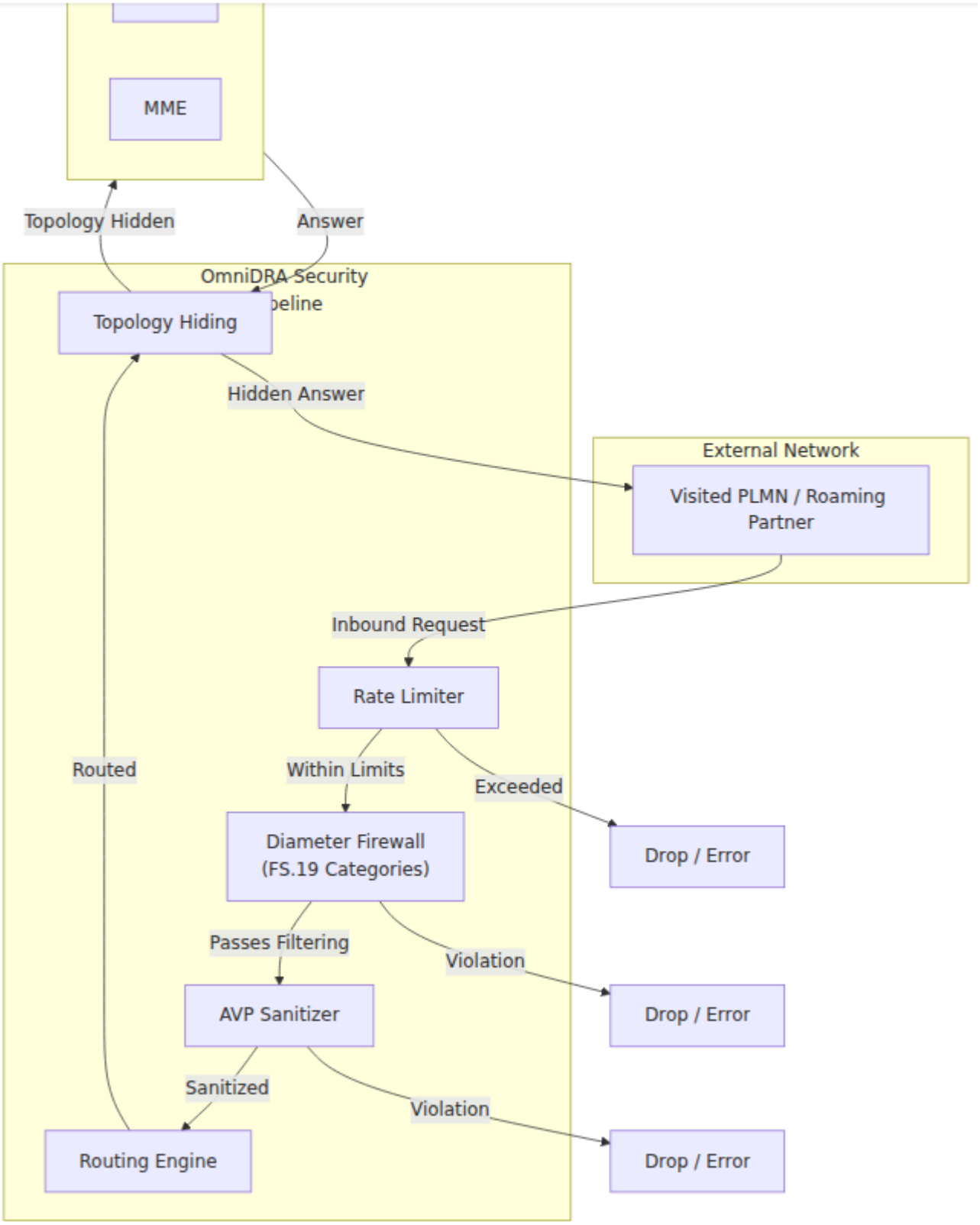
- RFC 4960 - Stream Control Transmission Protocol (SCTP)
- RFC 6733 - Diameter Base Protocol
- Configuration Reference

Diameter Security

OmniDRA provides a comprehensive suite of Diameter security modules. They align 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.

You configure each module independently. You can enable or disable each module without an effect on the others. Each module 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. If a stage rejects a message, the DRA never forwards it to the later 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

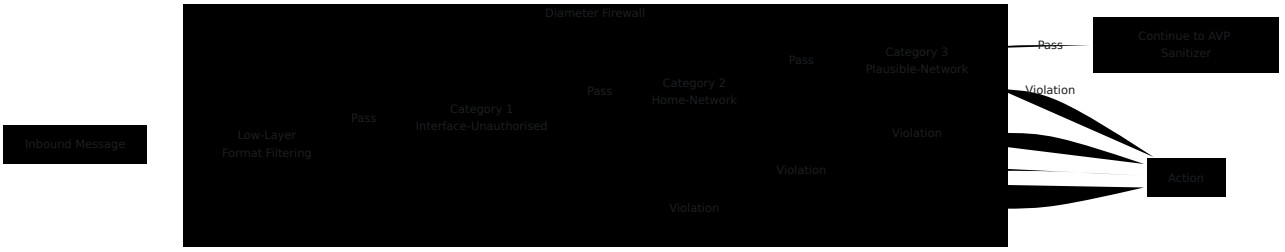
All security modules support configurable actions when they detect a violation. You configure the actions per-module, and per-category for the Diameter Firewall.

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`	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**. It also implements the protocol-agnostic packet categorisation from **FS.21 Section 7**. The firewall evaluates messages 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. It does not need 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 (for example Origin-Host, Origin-Realm). Prevents AVP doubling evasion attacks.
Session-Id Ordering	Section 3.3.4	Confirms 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 an answer message that contains a Destination-Host AVP. This is a protocol violation that may indicate filter evasion.
Mandatory AVP Validation	Section 3.3.5	Confirms that the required AVPs are present for each command code. For example, a 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 accepts only authorised Diameter Application IDs and Command Codes on each interconnect interface. This prevents external access to internal-only interfaces. For example, it blocks Sh commands over an S6a interface. It also enforces that a roaming partner sends only the message types that its roaming agreement covers.

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

You can configure whitelists globally, which applies them to all peers, or per-peer. A per-peer whitelist lets different roaming partners 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

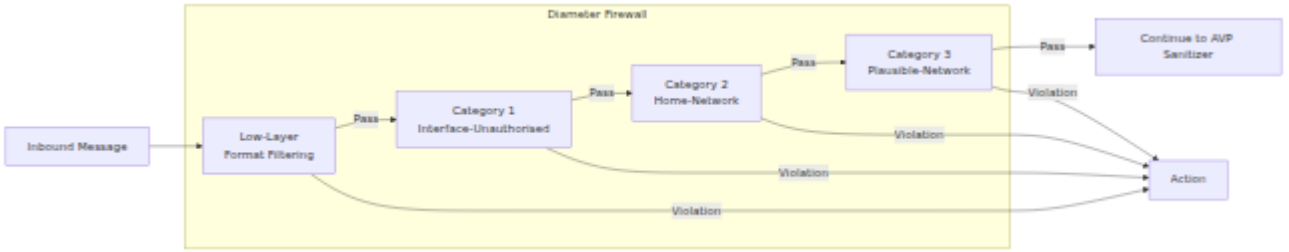
GSMA Reference: FS.19 Section 3.3.6, FS.21 Section 7.2.2

Category 2 filtering protects home subscribers against messages that target them from the interconnect. The firewall inspects messages on protected command codes for subscriber identity (IMSI in the User-Name AVP, MSISDN in AVP 701). If the identity matches a home subscriber prefix, the firewall rejects the message. Legitimate traffic for home subscribers originates 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 that target 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. It tracks the last-seen network for each subscriber (IMSI). When an Update-Location-Request arrives from a different visited network than the previous one, the firewall compares the elapsed time against a configurable threshold. A realm change that occurs faster than is physically possible indicates a potential attack, such as a spoofed location update.

This module maintains stateful per-IMSI tracking in an ETS table. It cleans up stale entries automatically. It removes entries older than 24 hours 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 a realm change that occurs 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

Enable the Diameter Firewall at the top level. Configure each filtering layer independently. Combine the config snippets from each category section above 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 that t enabled. For the `[DRA.Router.Mo` Category 3 state
`home_imsi_prefixes`	List	No	`[]`	List of IMSI prefi Category 2 filter
`home_msisdn_prefixes`	List	No	`[]`	List of MSISDN p Category 2 filter

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 the fir a Category 1 violation. The responds 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 message targets a home subscriber from the 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]`	The firewall detected and blocked a low-layer format violation
`[:diameter, :firewall, :category_1, :block]`	The firewall detected and blocked a Category 1 violation
`[:diameter, :firewall, :category_2, :block]`	The firewall detected and blocked a Category 2 violation
`[:diameter, :firewall, :category_3, :block]`	The firewall detected and blocked a Category 3 violation
`[:diameter, :firewall, :pass, :count]`	The message passed all firewall checks

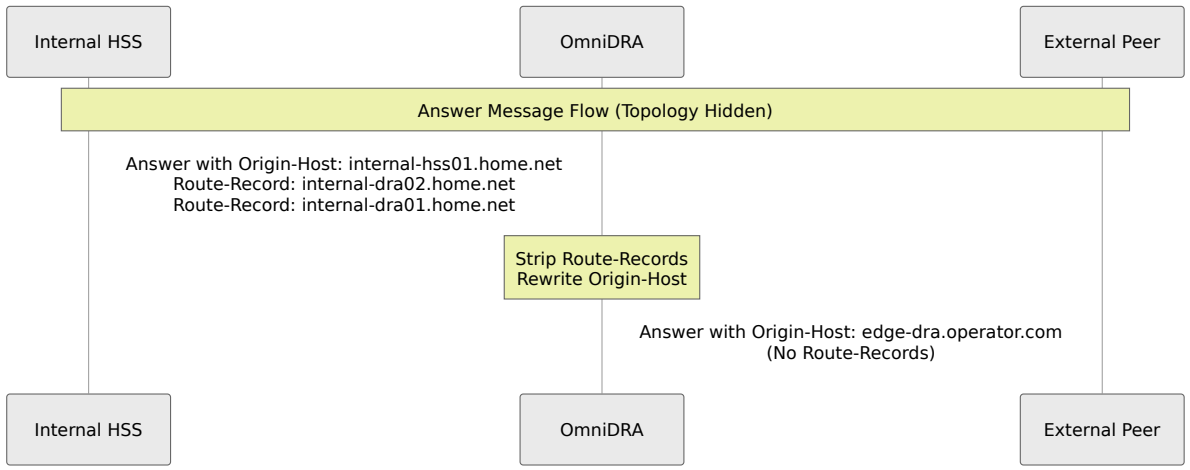
All events include this metadata: `origin_host`, `application_id`, `command_code`, `application_name`, `command_name`, and `reason`.

Topology Hiding

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

The Topology Hiding module prevents exposure of the internal network topology to external peers. When Diameter messages traverse multiple internal nodes, they accumulate Route-Record AVPs. They also carry Origin-Host/Origin-Realm values that reveal internal hostnames, network structure, and node counts. An attacker can use this information to map the internal network for targeted attacks.

Topology hiding operates on the **outbound path**. It runs after the DRA makes routing decisions and before it forwards messages 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 the 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 the DRA forwards them 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 that the module treats as external. The module

Parameter	Type	Required	Default	Description
				applies topology hiding only to messages that are 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`). The module uses a string value as-is.

Telemetry Events

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

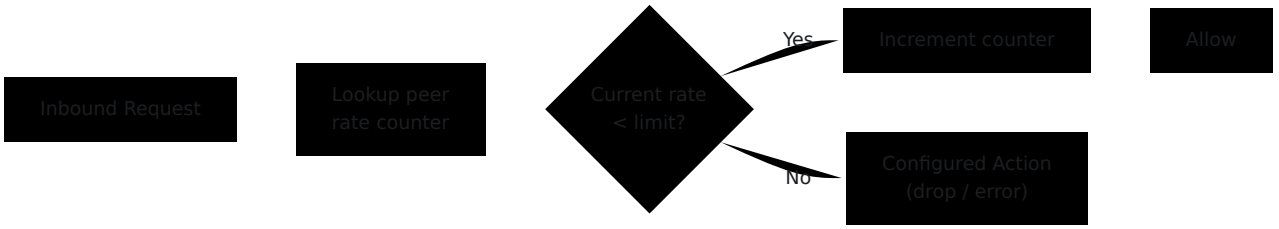
Rate Limiter

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

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

The Rate Limiter tracks rate limits per-peer with 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 sta `[D` so ru
`default_max_requests_per_second`	Integer	No	`1000`	De se
`default_action`	Action	No	`{:error, 3004}`	De its res
`peer_limits`	Map	No	`%{}`	Ma co the

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]`	The module rate limited a message. The measurements include `current_rate` and `limit`.
`[:diameter, :rate_limiter, :allowed]`	A message was within the 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. These recommendations handle 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 that use 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 the modu a nesting violation. stripping the offen and does the mess
`allowed_vendor_ids`	List	No	`[0, 10415, 13019, 5535]`	List of ve permitted interconn module s from othe
`strip_unknown_vendor_avps`	Boolean	No	`true`	Enable st AVPs from that are n `allowed_` list.
`max_avp_nesting_depth`	Integer	No	`10`	Maximum depth of AVP nesti configure applies to message exceeds t

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]`	The module stripped one or more AVPs from unknown vendors
`[:diameter, :avp_sanitizer, :nesting_depth, :violation]`	A message exceeded the maximum AVP nesting depth
`[:diameter, :avp_sanitizer, :pass, :count]`	The message passed all sanitization checks

Deployment Recommendations

Recommended Enablement Order

When you deploy the security modules for the first time, enable them one at a time. This avoids disruption to live traffic:

1. **Rate Limiter** - Start with generous limits and monitor traffic patterns. Tighten the limits after you understand the baseline rates.

2. **AVP Sanitizer** - Low risk of false positives. Enable the 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 the protocol conformance checks.
5. **Diameter Firewall (Category 2)** - Configure the home IMSI/MSISDN prefixes.
6. **Topology Hiding** - Enable Route-Record stripping first, then Origin-Host rewriting.
7. **Diameter Firewall (Category 3)** - Enable it with the `:log_only` action. Observe location patterns before you enforce.

Defence in Depth

These modules implement the **defence in depth** principle from 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. The DRA applies it when no Advanced Routing rule matches an incoming request. It follows the priority-based routing mechanism defined in the Diameter Base Protocol (RFC 6733). It selects a destination from the 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. The rule can route to named peers, route by `Destination-Host`, generate an error answer, or drop the request. 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 stages. When enabled, Rx Session Binding redirects an Rx request to the PCRF that holds its session. It only overrides a relay decision. A `:drop` outcome, an error answer, and an explicit `Destination-Host` all stand.

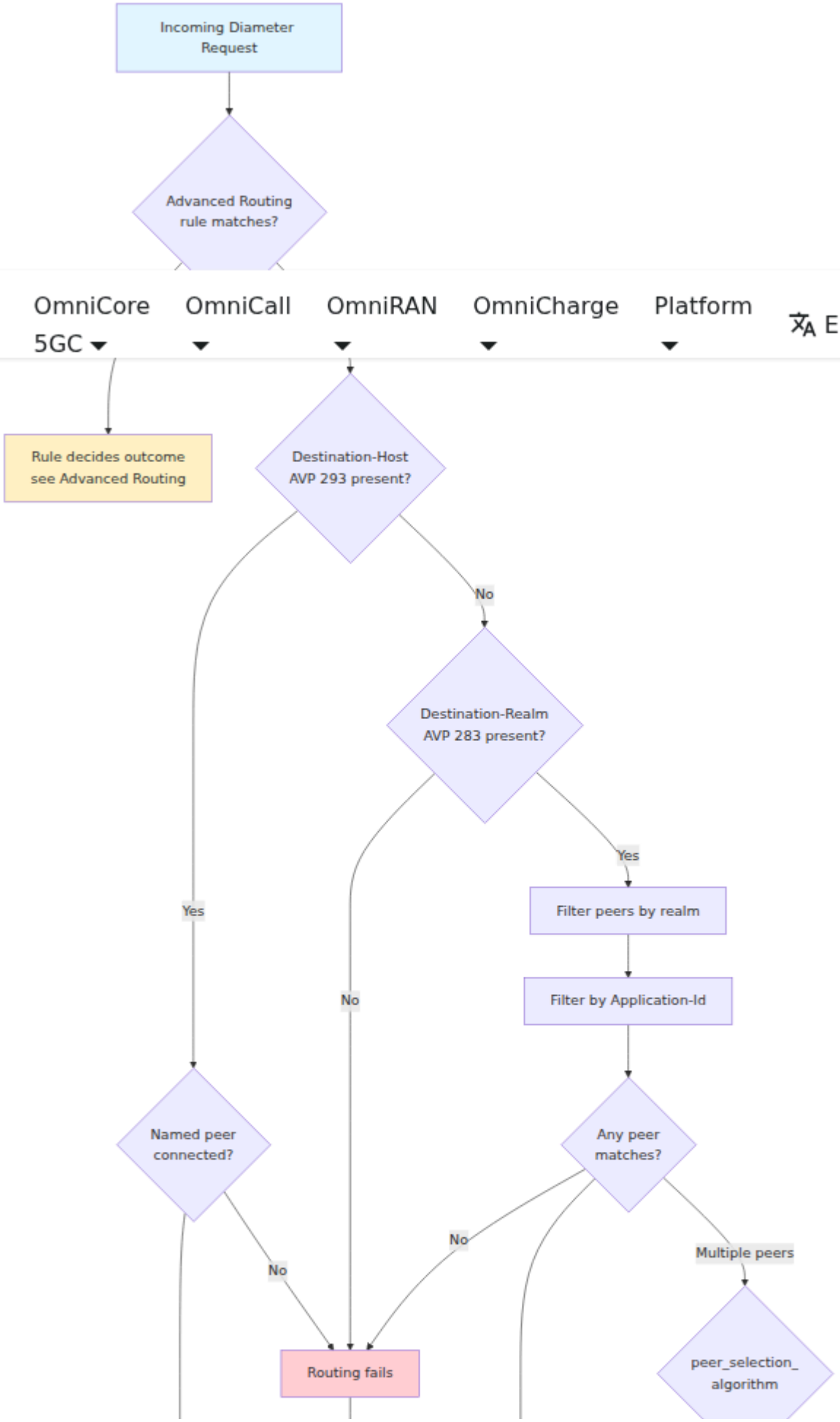
Advanced Routing describes the content-based overrides. These override routing on AVP values such as IMSI patterns, realm translation, and originating-peer decisions. That page also details **session affinity** through the `Destination-Host` AVP.

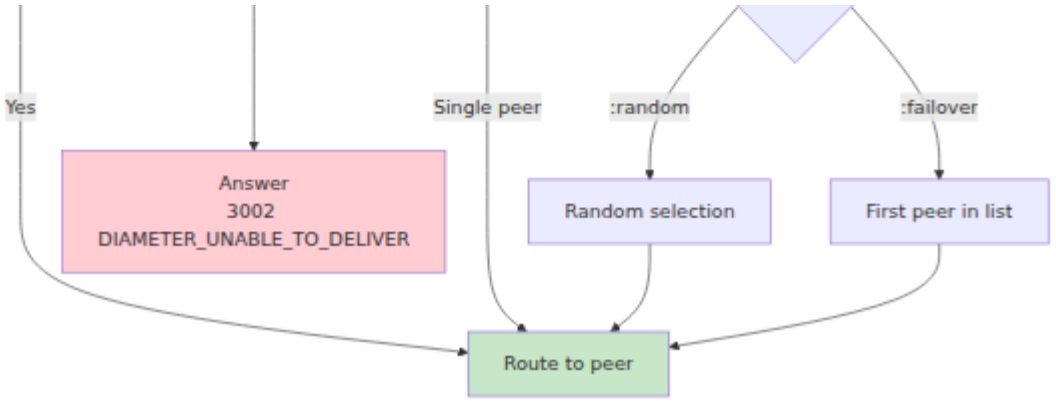
Routing Decision Precedence

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

1. **`Destination-Host` AVP (293)** - This is the highest-priority mechanism. If the request carries a `Destination-Host` AVP, the DRA routes directly to that host. This gives explicit, host-level routing control. If the named peer is not connected, routing fails.
2. **`Destination-Realm` AVP (283) + `Application-Id`** - If the `Destination-Host` is absent, the DRA selects a connected peer that advertises support for the target **realm**. The DRA then filters the candidate peers by the message's **Application-Id**. Only peers that advertise support for that application remain. The peers advertise this support during Capabilities Exchange. 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 DRA cannot deliver the request. The DRA returns a `3002 DIAMETER_UNABLE_TO_DELIVER` answer.

The following flowchart shows the full decision path. It includes the point at which Advanced Routing can 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. The DRA automatically excludes disconnected or down peers from realm and application filtering. 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. You do not configure it 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 for when multiple connected peers match a request. `:random` selects a random peer from the matching set. `:failover` always selects the first peer in the configured list. It uses the remaining peers only when the first is unavailable.

- **`:random`** - Spreads load evenly across all matching peers. Use this when peers are interchangeable and equal-cost.
- **`:failover`** - Prefers a single primary peer. It treats the ordered peer list as a priority sequence. Use this for active/standby designs. A specific peer carries traffic while it stays connected.

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

When No Peer Is Available

The precedence evaluation can fail to identify a reachable destination. This happens when the `Destination-Host` peer is disconnected, when no connected peer advertises the target realm, when no matching peer supports the Application-Id, or when the request lacks the AVPs needed to make a decision. In each of these cases, the DRA cannot forward the request.

The DRA then returns an answer with `Result-Code` set to `3002 DIAMETER_UNABLE_TO_DELIVER`, as defined in RFC 6733 §7.1.5. This tells the originating peer that the DRA received the request but could not deliver it to a suitable next hop. The originator can then apply its own retry or failover logic.

Reference

Result Codes

Code	Name	Description	Reference
2001	DIAMETER_SUCCESS	The DRA completed the request successfully	RFC 6733 §7.1.1
3002	DIAMETER_UNABLE_TO_DELIVER	The DRA could not deliver the request 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 the 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 serves each subscriber. It observes the Diameter signalling traffic that passes through the DRA. It uses the learned bindings to route later requests, such as location service queries, directly to the correct serving node. It needs no static routing rules.

This is useful for SLg/SLh location service requests. The GMLC needs to reach the serving MME for a given subscriber but does not know which MME that is.

3GPP TS 29.172 and 3GPP TS 29.173 describe the SLF concept for location services. 3GPP TS 29.272 defines the subscriber registration.

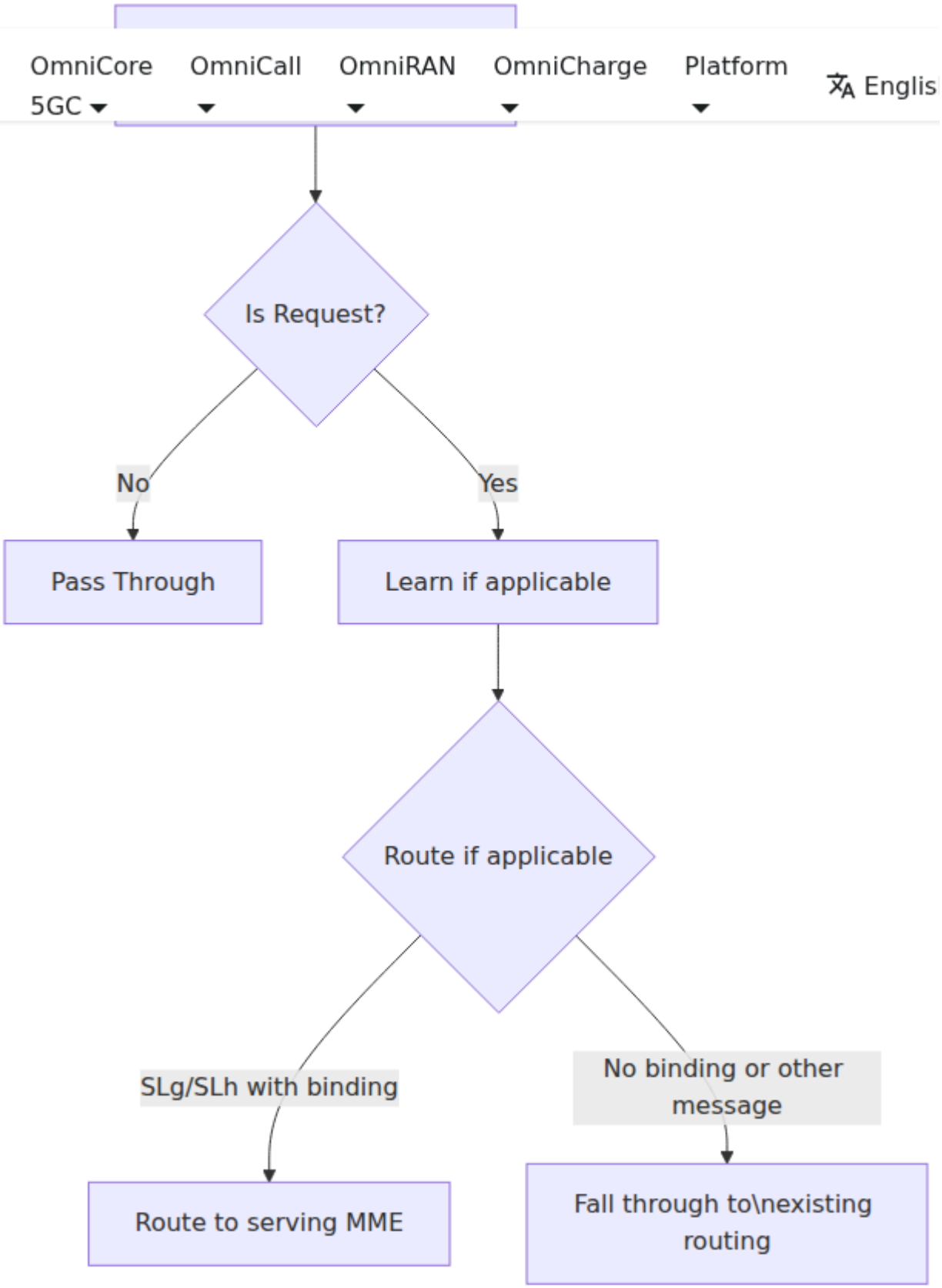
How It Works

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

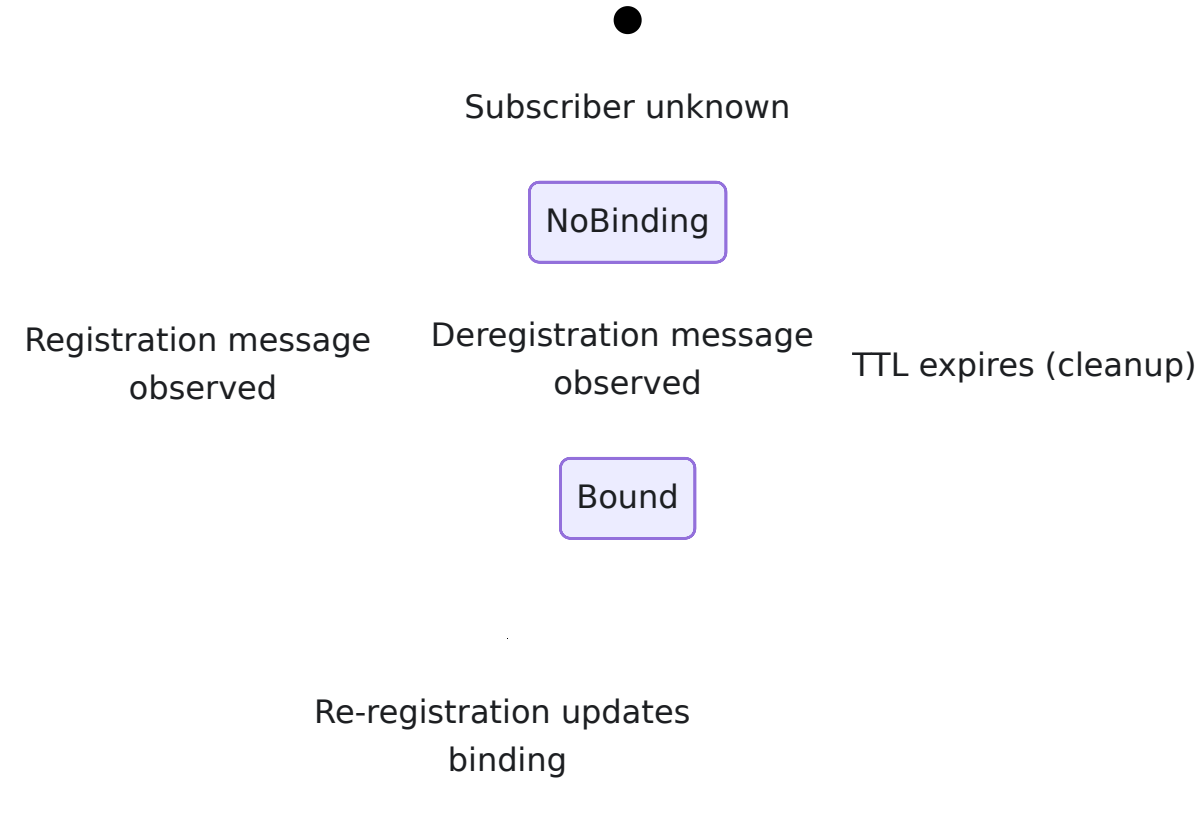
During **learning**, the module observes registration and deregistration messages that flow through the DRA. When a subscriber registers (for example through an S6a Update-Location-Request), the module records which network element now serves that subscriber.

During **routing**, a request can arrive that must reach a subscriber's serving node (for example 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.

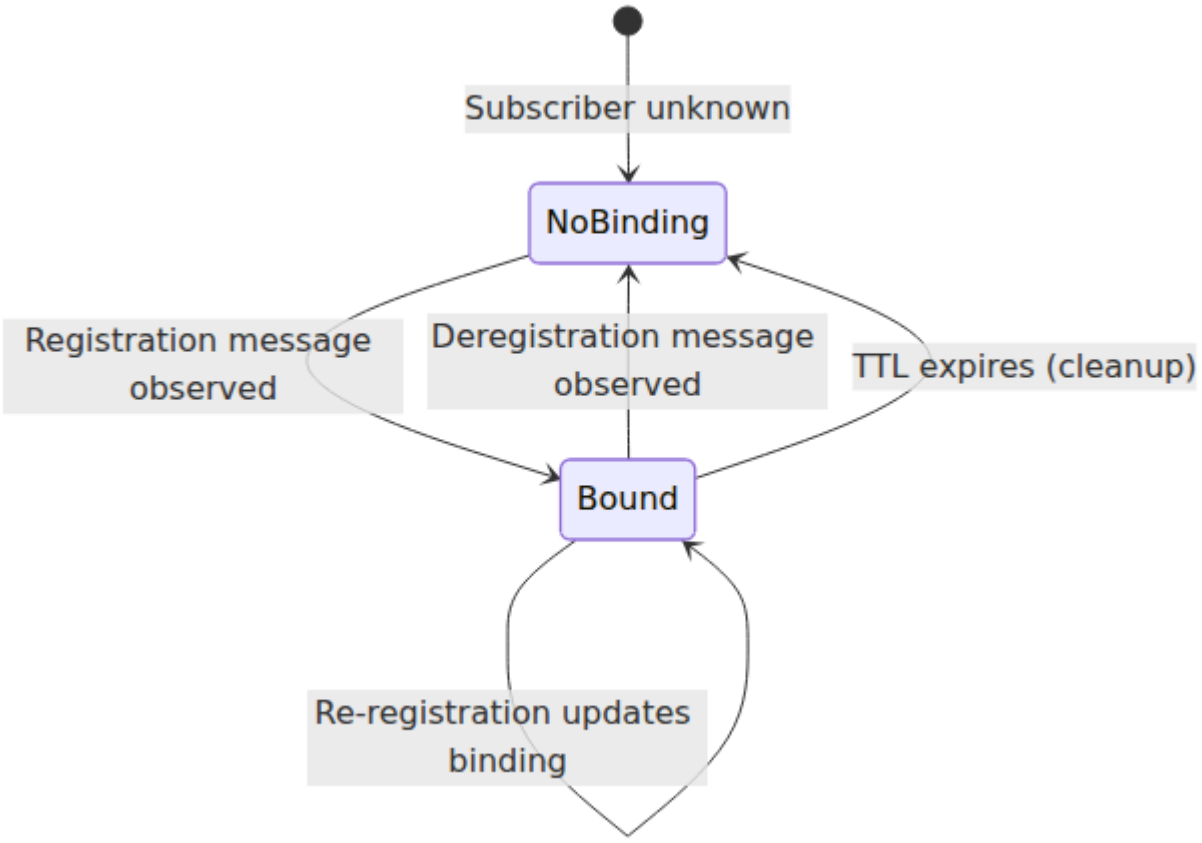

Is Request?
No
Yes
Pass Through
Learn if applicable
Route if applicable
SLg/SLh with binding
No binding or other message
Route to serving MME
Fall through to\nexisting routing

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 module removes the last binding for a subscriber, it deletes the entire subscriber record from the table.

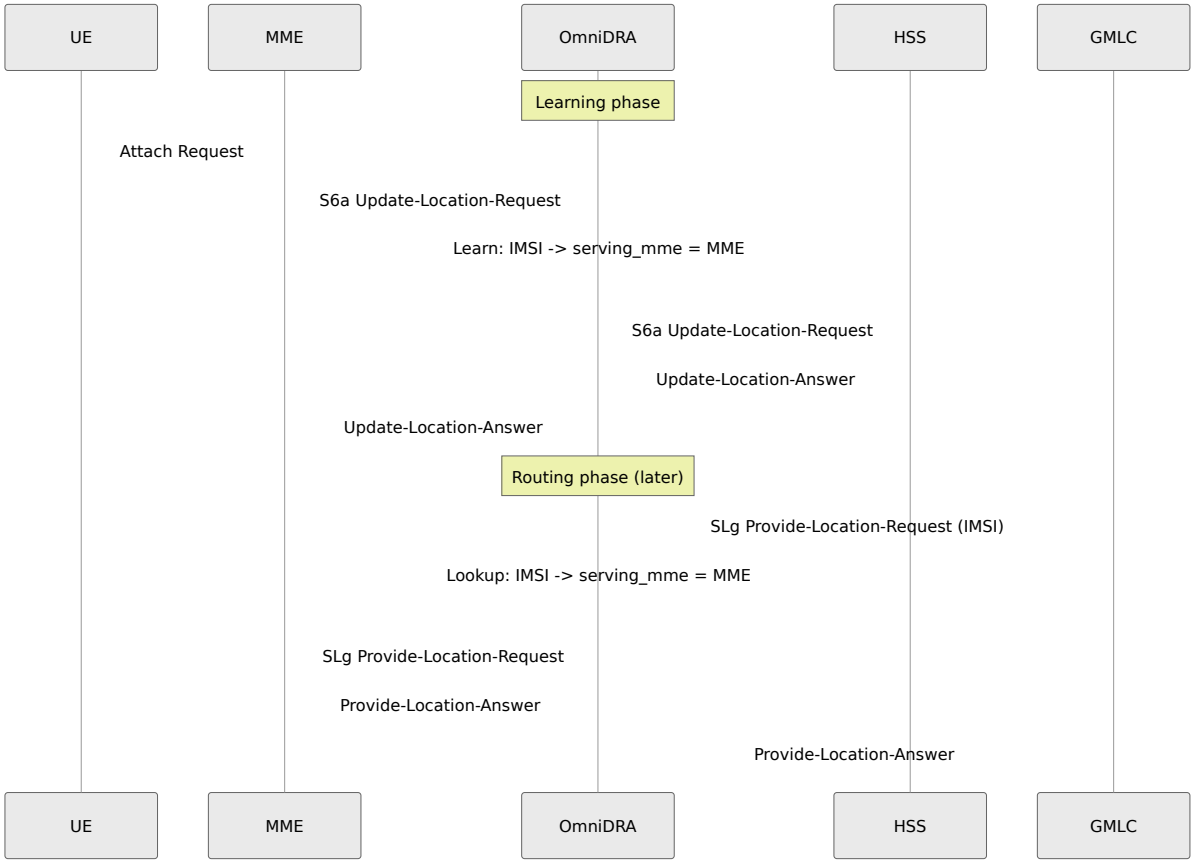
Routing: Messages That Use Bindings

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

The DRA relays a request routed by a learned binding to the serving node. It uses 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. The SLF route takes precedence.

Configuration

Configure the module 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 learns no binding
`children`	List	No	`[]`	Processes that th when enabled. Se `[DRA.Router.Mod` so the binding-tal
`binding_ttl_seconds`	Integer	No	`86400`	Time in seconds b binding expires. C The default is 24 half the TTL inter 60 seconds.

Configuration Examples

Standard Deployment

Use this 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. The DRA automatically routes SLg and SLh requests to the correct serving MME. The module refreshes a binding whenever it observes a new registration message for the same subscriber.

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

High-Churn Environment

Use this for networks with frequent subscriber mobility. Stale bindings must expire quickly.

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

How it works: Bindings expire after 1 hour. Use this when subscribers move often between MMEs. Stale bindings would then 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**: The DRA increments this counter each time it creates or updates

a subscriber binding. **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**: The DRA increments this counter each time a deregistration message removes a subscriber binding. **Labels**:

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

Routed Requests

Metric: `diameter.subscriber_lookup.route.count` **Type**: Counter **Description**: The DRA increments this counter each time it routes an SLg/SLh request with a learned binding. **Labels**:

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

Troubleshooting

Location Requests Not Being Routed to Serving MME

Symptoms: The DRA does not route SLg Provide-Location-Requests or SLh LCS-Routing-Info-Requests to the expected MME. They fall through to default routing instead.

Possible causes:

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

Resolution:

1. Confirm the module is enabled in the configuration.
2. Check that S6a ULR traffic for the subscriber flows through this DRA. The module learns bindings only from observed traffic.
3. Confirm that `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 the supervisor.
- Messages do not contain a valid IMSI in the expected AVP.

Resolution:

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

Stale Bindings Causing Misrouted Requests

Symptoms: The DRA routes location requests to an MME that no longer serves 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. Confirm that 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

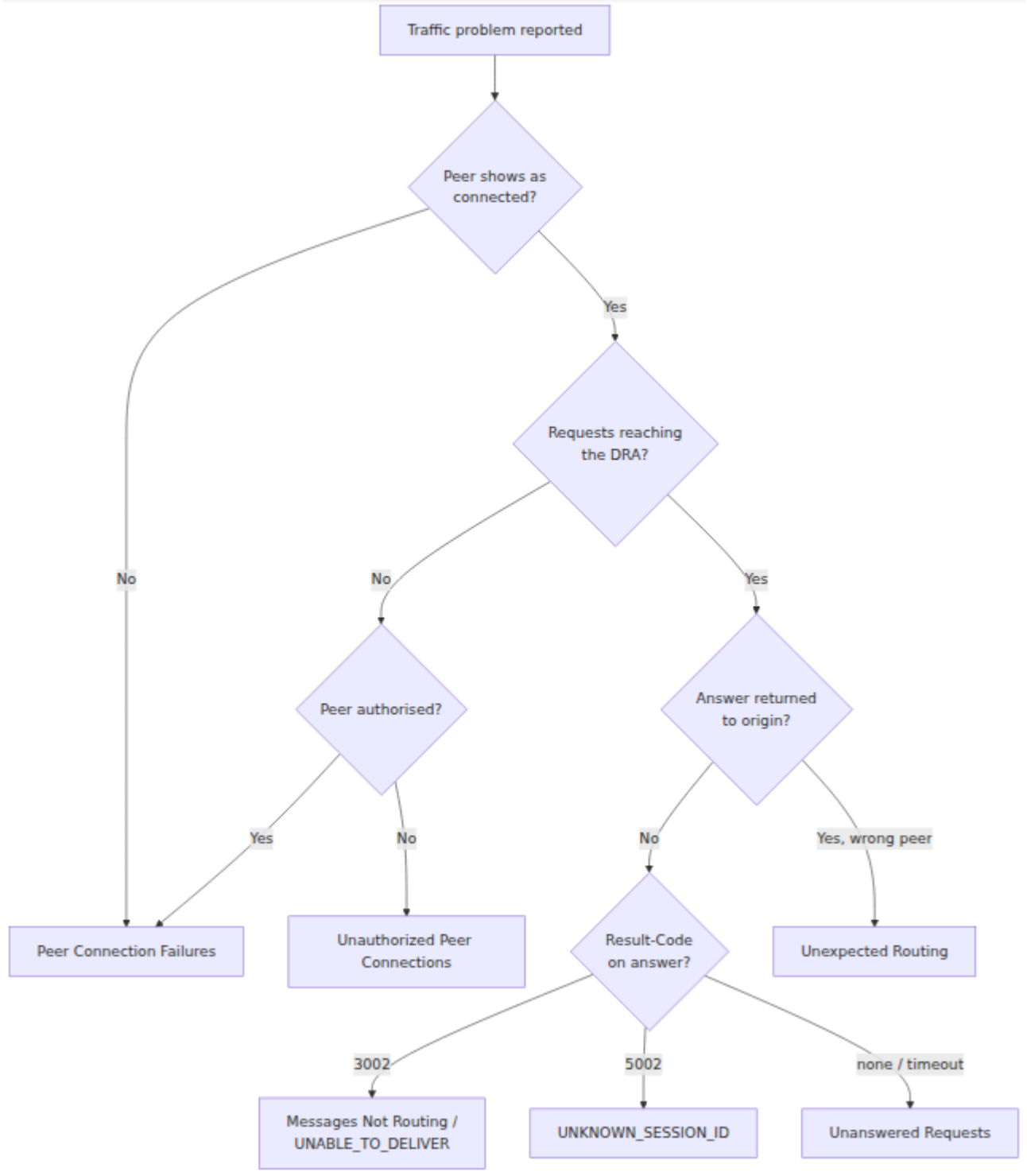
Binding Type	Learned From	Used By	Description
`serving_mme`	S6a ULR	SLg PLR, SLh RIR	The MME currently serving the subscriber
`serving_pgw`	Gx CCR-I	-	The PGW handling the subscriber's session (reserved for future routing)
`serving_cscf`	Cx SAR (Registration)	-	The S-CSCF serving the subscriber's IMS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

Troubleshooting

This guide covers the most common issues in OmniDRA operation. Each entry lists the symptoms that an operator observes, the likely causes, and an ordered set of steps. Use the steps to isolate and resolve the problem. For the metric names in this guide, see Metrics.

Diagnosis Decision Tree

Use this flow to find where a problem sits before you go to 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 applies to an answer.
- The rule uses the wrong filter type for the module.

Resolution:

1. Check 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 against real AVP values before you deploy them.
4. Confirm the message type falls within the rule's `match` scope.
5. Review the available filter types in Advanced Routing. Confirm you use the correct one for the module.

Unexpected Routing

Symptoms: The DRA forwards requests, but to a different peer than intended. Alternatively, the DRA uses standard routing when you expected a rule 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 the rule order. Place more specific rules before broader ones. The first matching rule wins. See Advanced Routing.
2. Confirm that peer names match the configuration exactly and that the target peer is connected (`diameter_peer_status` reports `1`).
3. Check for conflicting rules with overlapping filters. Reorder or tighten them.
4. Confirm the fall-through behaviour 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 transform targets an AVP code that is incorrect for the use case.
- An `edit` action ran 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 message never reaches the transform.
- The packet-type filter (`:request` vs `:answer`) excludes the message.
- The action does not support 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, confirm the AVP already exists in the message. Use `add` if the AVP may be absent.

3. Confirm that the rule's filters match the intended message flow.
4. Confirm that any packet-type filter (`:request` vs `:answer`) is correct for the direction you transform.
5. Confirm the action supports the packet type. `edit` works on requests only.

Peer Connectivity

Peer Connection Failures

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

Possible causes:

- A firewall blocks the configured `listen_port` (default `3868`) or the peer's port.
- The peer configuration has an incorrect `ip`, `port`, or `transport`.
- 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`. The identities must match exactly.
- `initiate_connection` is set the same way on both ends, so neither side dials, or both dial.
- 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 that allow SCTP, not just TCP.
2. Confirm the peer's `ip`, `port`, and `transport` values match the peer's actual endpoint exactly.

3. Confirm 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 the `initiate_connection` roles. Exactly one side must initiate. Set `initiate_connection: true` on the side that dials (typically the DRA toward an HSS or PCRF). Set `false` on the side that waits (typically toward an MME, SGSN, or P-GW).
6. Confirm the peer advertises at least one application that the DRA serves during CER/CEA. A peer with no common application is not 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, the DRA dispatches requests to it, and they 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 reconnects repeatedly (flapping).
- The application-level settle time is too short for this peer, so the DRA routes traffic before the peer is ready.
- 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 the DRA routes traffic to it. A peer that disconnects inside

this window never enters the routable set. This keeps traffic 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 stay up before the DRA routes traffic to it. `0` routes immediately after CER/CEA completes. If the peer disconnects during the window, the DRA never marks it routable. This keeps traffic 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 it unset unless you have a specific reason to change the liveness cadence.

3. `diameter_peer_status` reflects actual routability. A peer inside its settle window still reports as not-yet-routable in metrics. Use this to confirm the settle behaviour works as 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 after the application-level settle time elapses. It does not wait out the RFC 3539 watchdog's implicit reconnect delay. The settle time above is the single, explicit control over how long the DRA holds a come-up.

Unauthorized Peer Connections

Symptoms: The DRA rejects a peer's connection. Alternatively, `diameter_peer_unauthorized_connection_count_total` increments with an unexpected `origin_host` / `peer_ip`.

Possible causes:

- The connecting peer is not 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 tries to connect.

Resolution:

1. Confirm whether to allow the peer. If so, add it to the `peers` list with its exact `host`, `realm`, `ip`, `port`, and `transport`.

2. Confirm the peer's advertised Origin-Host matches the configured `host` exactly. The DRA treats a mismatch as an unknown peer.

3. Keep `allow_undefined_peers_to_connect: false` in production and add peers explicitly. Enable it only in controlled environments where any peer may connect.

4. Keep `log_unauthorized_peer_connection_attempts: true` so the DRA logs rejected attempts 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. Alternatively, 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 in the peer's `additional_ips` list.

- Only one endpoint is configured for multihoming, so the DRA does not achieve full redundancy.
- Kernel SCTP parameters govern the path failover timing, and it differs from expectations.

Resolution:

1. Confirm the SCTP kernel module is loaded (the `lksctp-tools` package on Linux).
2. Confirm firewall rules allow SCTP traffic on every configured IP, not only TCP.
3. Confirm 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, the DRA uses only `ip`, because TCP does not support multihoming.
5. Configure both endpoints for multihoming for full path redundancy.
6. If failover timing is not as expected, review the 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 the DRA answers them with Result-Code `3002` (DIAMETER_UNABLE_TO_DELIVER). Alternatively, the DRA does not forward them 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 it 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.
- No peer serves the requested realm. This yields `3003` (DIAMETER_REALM_NOT_SERVED), not `3002`.

Resolution:

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

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

3. Confirm the peer advertises the application in the request during CER/CEA. The DRA excludes peers that do not advertise the application from selection.

4. If an advanced routing rule targets a specific peer, confirm that peer is connected. The DRA skips disconnected peers. See Advanced Routing.

5. Distinguish `3002` from `3003`. `3002` means the DRA could not select a reachable peer. `3003` means no peer serves the requested realm. Correct the 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: The DRA forwards requests 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. This is 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. This shows 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 exceeds the default. A higher value 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 DRA treats the request/answer pair as 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 the peer rejects an update (CCR-U) or termination (CCR-T) even though the initial (CCR-I) succeeded.

Possible causes:

- The DRA routed a mid-session request to a different peer than the one that holds the session state.
- The peer that holds the session restarted and lost its session table.
- The DRA routed to a peer that reconnected before it restored its state. This is a settle-window interaction.
- Load balancing spread requests for one Session-Id across multiple peers.

Resolution:

1. Confirm that all requests with 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. Affinity is then explicit and does not rely on load balancing.

3. Check whether the target peer restarted around the time the `5002` answers began. A peer that lost its session table rejects in-flight sessions. The client must re-establish those sessions.

4. If a peer recently reconnected, confirm `peer_settle_time_ms` is large enough that the DRA routes traffic only after the peer is 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	The DRA completed the request successfully.	RFC 6733 §7.1.1
3002	DIAMETER_UNABLE_TO_DELIVER	The DRA could not select a reachable peer 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. This is a lost or misrouted stateful session.	RFC 6733 §7.1.5
5012	DIAMETER_UNABLE_TO_COMPLY	The DRA cannot process the request 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.