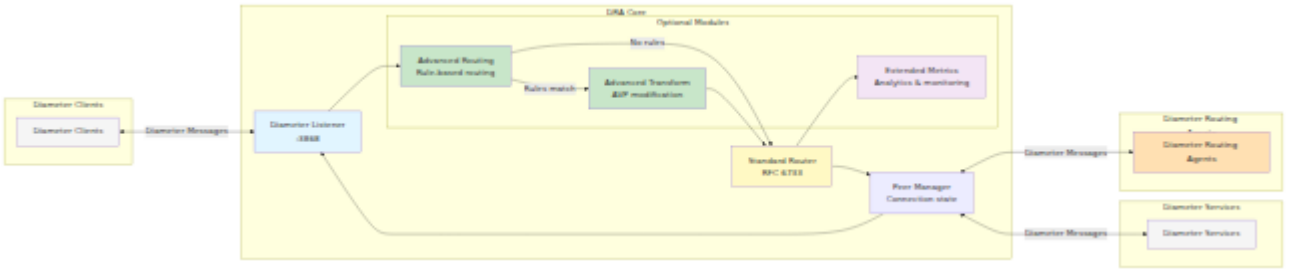
DRA Operations Guide

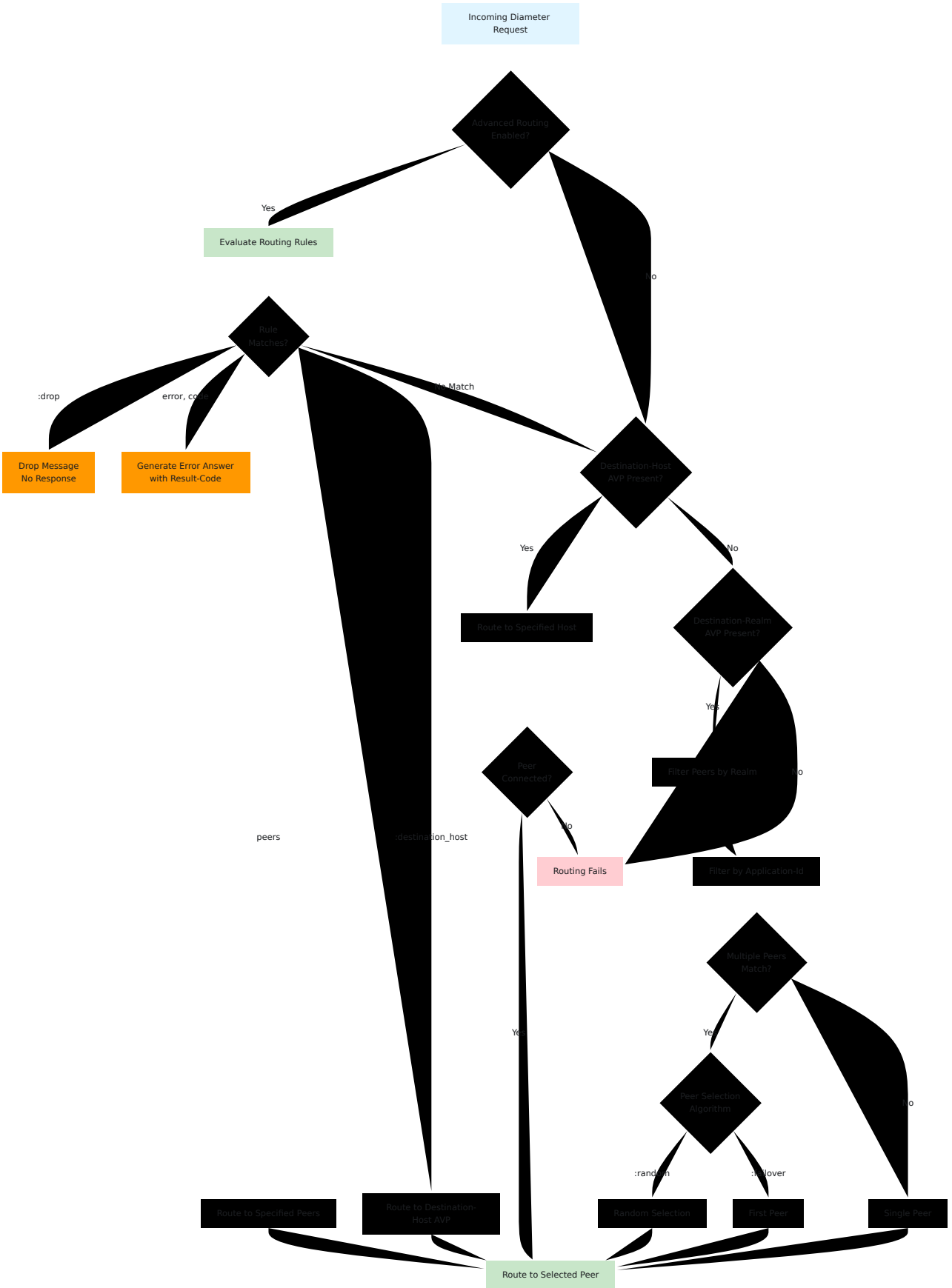
Table of Contents

DRA Architecture Overview



Standard Diameter Routing

Without the Advanced Routing or Advanced Transform modules, the DRA performs standard Diameter routing based on the Diameter Base Protocol (RFC 6733):


Incoming Diameter Request
Advanced Routing Enabled?
Yes
Evaluate Routing Rules
No
Rule Matches?
:drop
error, code
Drop Message No Response
Generate Error Answer with Result-Code
Destination-Host AVP Present?
Yes
No
Route to Specified Host
Destination-Realm AVP Present?
Yes
Filter Peers by Realm
No
Peer Connected?
No
Routing Fails
Filter by Application-Id
peers
:destination_host
Multiple Peers Match?
Yes
Peer Selection Algorithm
:random
:failover
Random Selection
First Peer
Single Peer
Route to Specified Peers
Route to Destination-Host AVP
Route to Selected Peer

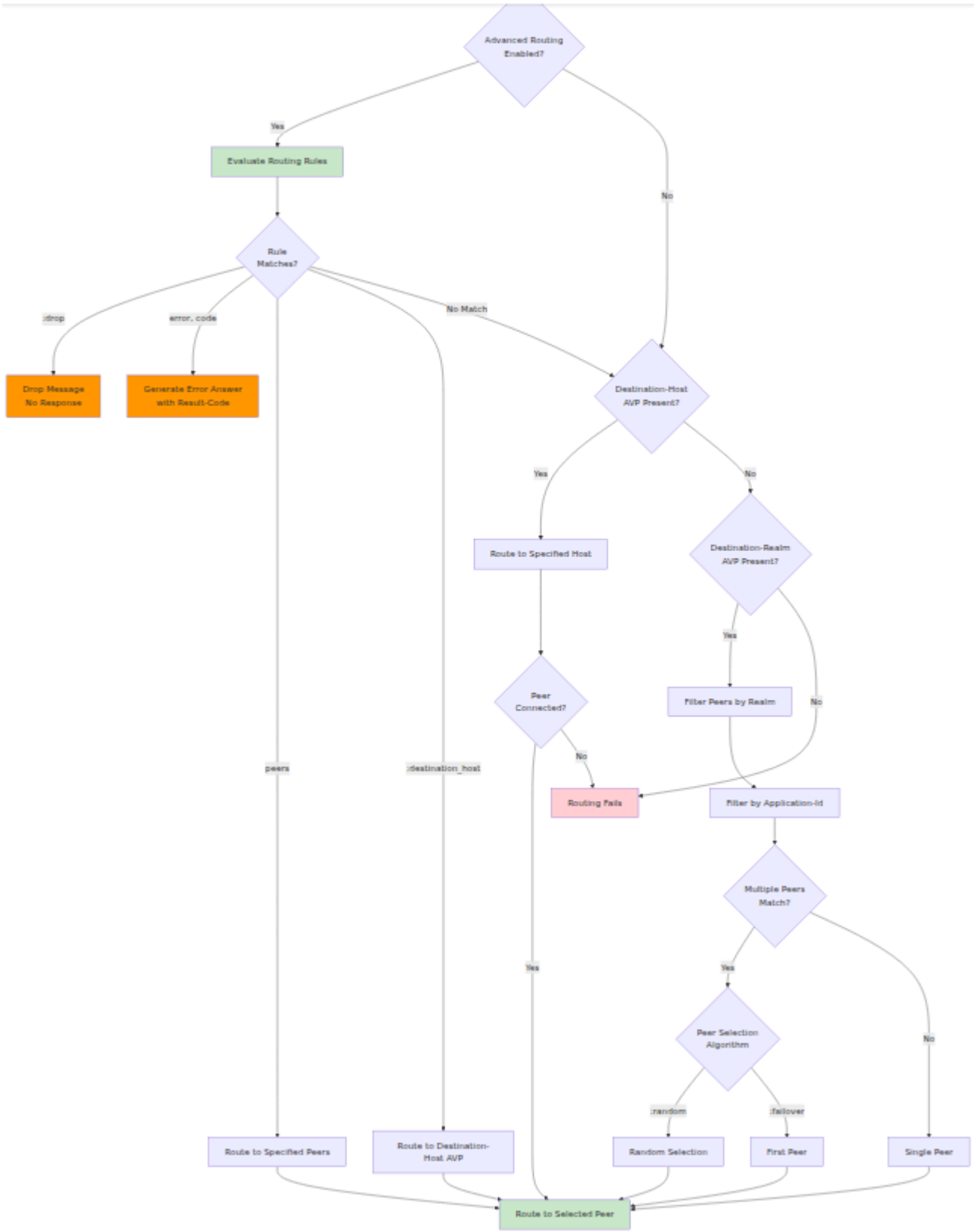
Request Routing

The DRA routes request messages using a priority-based mechanism defined in RFC 6733 Section 6.1:

1. **Destination-Host AVP (293)** - If present, the DRA routes directly to the specified peer
    - This is the highest priority routing mechanism
    - If the peer is not connected, routing fails
    - Provides explicit, host-level routing control
2. **Destination-Realm AVP (283)** - If Destination-Host is absent, routes based on realm
    - The DRA selects a connected peer that advertises support for the target realm
    - Load balancing is applied when multiple peers match the realm
    - Realm-based routing allows flexibility across multiple hosts
3. **Application-Id** - Peers are filtered by supported Diameter applications
    - Only peers advertising support for the message's Application-Id are considered
    - Based on Capabilities Exchange (CER/CEA) during peer connection establishment
    - See Common 3GPP Application IDs for reference

Answer Routing

Answer packets use a fundamentally different routing mechanism than requests:



- **Session-based routing**: Answer packets always follow the reverse path of the request

- **End-to-End ID preservation**: The End-to-End Identifier remains unchanged across all hops
- **Hop-by-Hop routing**: The DRA uses the Hop-by-Hop Identifier to maintain routing state (changes at each hop)
- **No rule evaluation**: The DRA does not evaluate routing rules or AVP contents for answers
- **Stateful correlation**: Internal routing tables track which peer sent each request

Why answers are not routed by advanced modules:

- Answer routing is deterministic and must return to the originating peer
- The Diameter protocol requires answers to follow the established request path
- Routing decisions for answers are made based on the original request context, not answer content
- This ensures proper session management and prevents routing loops

See RFC 6733 Section 6.2 for answer message routing details.

Peer Selection

When multiple peers match the routing criteria, the configured `peer_selection_algorithm` determines selection:

- `:random` - Randomly selects from available peers (default)
- `:failover` - Always selects the first peer in the list (priority-based)
- Peers must be in **connected state** to be selected
- Disconnected or down peers are automatically excluded

Limitations of Standard Routing

- No custom routing rules based on AVP values (e.g., IMSI patterns)
- No realm translation or AVP modification
- Cannot route based on originating peer
- Limited control over traffic distribution

The Advanced Routing and Advanced Transform modules extend this standard behavior with rule-based routing and packet manipulation capabilities.

Base DRA Configuration

The DRA requires base configuration defining its identity, network settings, and peer connections. This configuration establishes the foundation for all routing operations.

Configuration Structure

```
%{
  host: "dra01.example.com",
  realm: "example.com",
  listen_ip: "192.168.1.10",
  listen_port: 3868,
  service_name: :example_dra,
  product_name: "OmniDRA",
  vendor_id: 10415,
  request_timeout: 5000,
  peer_selection_algorithm: :random,
  allow_undefined_peers_to_connect: false,
  log_unauthorized_peer_connection_attempts: true,
  peers: [
    # Peer configurations...
  ]
}
```

DRA Identity Parameters

Parameter	Type	Description
`host`	String	The DRA's Diameter Identity (fully qualified domain name)
`realm`	String	The DRA's Diameter realm
`product_name`	String	Product name advertised in CER/CEA messages
`vendor_id`	Integer	Vendor-ID as defined in RFC 6733 Section 5.3.3 (10415 = 3GPP)

Network Settings

Parameter	Type	Description
`listen_ip`	String or List	IP address(es) the DRA listens on. For SCTP multihoming, use a list of IP strings (see SCTP Multihoming)
`listen_port`	Integer	TCP/SCTP port for Diameter connections (standard: 3868)
`service_name`	Atom	Internal Erlang service identifier
`request_timeout`	Integer	Timeout in milliseconds for request/answer pairs (default: 5000)

Peer Selection Settings

Parameter	Type	Description
`peer_selection_algorithm`	Atom	Load balancing algorithm: `:random` (random selection) or `:failover` (first peer priority)
`allow_undefined_peers_to_connect`	Boolean	Allow connections from peers not in configuration (default: `false`)
`log_unauthorized_peer_connection_attempts`	Boolean	Log connection attempts from unauthorized peers

Peer Configuration

Each peer in the `peers` list defines a Diameter connection:

```
%{
  host: "mme01.operator.com",
  realm: "operator.com",
  ip: "192.168.1.20",
  port: 3868,
  transport: :diameter_tcp,
  tls: false,
  initiate_connection: false
}
```

Peer Parameters

Parameter	Type	Description
`host`	String	Peer's Diameter Identity (FQDN) - must match exactly for routing
`realm`	String	Peer's Diameter realm
`ip`	String	Peer's primary IP address for connection (required)
`ips`	List	List of IP addresses for SCTP multihoming (optional, see SCTP Multihoming)
`port`	Integer	Peer's Diameter port (typically 3868)
`transport`	Atom	Transport protocol: `:diameter_tcp` or `:diameter_sctp`
`tls`	Boolean	Enable TLS encryption (if `true`, typically use port 3869)
`initiate_connection`	Boolean	`true`: DRA connects to peer, `false`: DRA waits for peer to connect

Connection Modes

Initiate Connection (`initiate_connection: true`)

- DRA acts as Diameter client
- DRA initiates TCP/SCTP connection to peer
- Used for connecting to HSS, PCRF, or other backend systems
- DRA will retry connections if peer is unreachable

Accept Connection (`initiate_connection: false`)

- DRA acts as Diameter server
- DRA waits for peer to connect
- Used for MME, SGSN, P-GW connections
- Peer must be in configuration or `allow_undefined_peers_to_connect: true`

Configuration Example

```
%{
  host: "dra01.mvno.example.com",
  realm: "mvno.example.com",
  listen_ip: "10.100.1.10",
  listen_port: 3868,
  service_name: :mvno_dra,
  product_name: "OmniDRA",
  vendor_id: 10415,
  request_timeout: 5000,
  peer_selection_algorithm: :random,
  allow_undefined_peers_to_connect: false,
  log_unauthorized_peer_connection_attempts: true,
  peers: [
    # MME - waits for MME to connect
    %{
      host: "mme01.operator.example.com",
      realm: "operator.example.com",
      ip: "10.100.2.15",
      port: 3868,
      transport: :diameter_sctp,
      tls: false,
      initiate_connection: false
    },
    # HSS - DRA initiates connection
    %{
      host: "hss01.mvno.example.com",
      realm: "mvno.example.com",
      ip: "10.100.3.141",
      port: 3868,
      transport: :diameter_tcp,
      tls: false,
      initiate_connection: true
    },
    # PCRF with TLS - DRA initiates secure connection
    %{
      host: "pcrf01.mvno.example.com",
      realm: "mvno.example.com",
      ip: "10.100.3.22",
      port: 3869,
      transport: :diameter_tcp,
      tls: true,
```

```
      initiate_connection: true
    }
  ]
}
```

Important Notes

- **Hostname Matching**: Peer hostnames in Advanced Routing rules must exactly match the `host` value configured here (case-sensitive)
- **Capabilities Exchange**: On connection, peers exchange supported applications via CER/CEA messages
- **Application Support**: The DRA advertises all supported 3GPP applications (see Common 3GPP Application IDs)
- **Vendor-ID 10415**: Standard value for 3GPP applications
- **Request Timeout**: Affects Extended Metrics TTL (timeout + 5 seconds)
- **Peer Selection**: When multiple peers match routing criteria, `peer_selection_algorithm` determines which is chosen

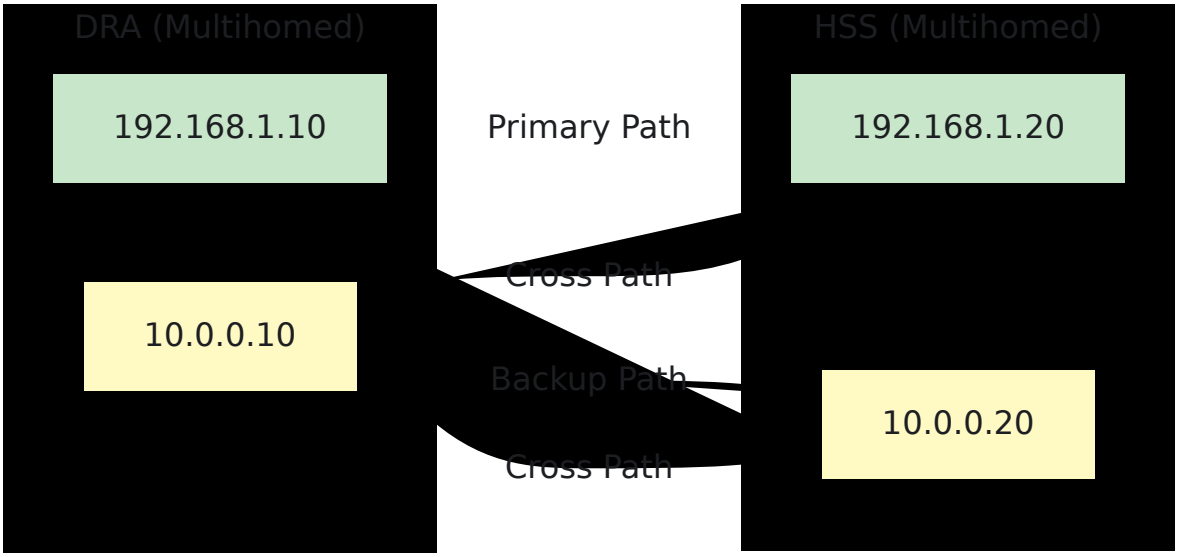
Security Considerations

- Set `allow_undefined_peers_to_connect: false` in production
- Enable `log_unauthorized_peer_connection_attempts: true` for security monitoring
- Ensure firewall rules match `listen_ip` and `listen_port` settings
- Validate peer certificates when using TLS

SCTP Multihoming

SCTP multihoming provides network redundancy by allowing endpoints to bind to multiple IP addresses. If the primary network path fails, SCTP automatically fails over to an alternative path without disrupting the Diameter session.

How It Works



- SCTP heartbeats monitor all network paths
- Automatic failover occurs if the primary path becomes unreachable
- No Diameter session disruption during path switchover
- The kernel handles path selection automatically

Configuration

DRA Listen Addresses

Configure multiple local IP addresses for the DRA to bind to:

```
%{
  # 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,
  ...
}
```

Notes:

- TCP transport uses only the first IP in the list
- SCTP transport binds to all specified IPs
- Single IP string format remains fully supported

Peer Configuration

Configure multiple remote IP addresses for peer connections:

```
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,
    tls: false,
    initiate_connection: true
  }
]
```

Notes:

- `ip` field is required for backward compatibility
- `ips` field is optional; if omitted, only `ip` is used
- For SCTP, include the primary IP in the `ips` list
- For TCP, only `ip` is used (TCP does not support multihoming)

Complete Example

```
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,
        tls: false,
        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,
        tls: false,
        initiate_connection: false
      }
    ]
  }
```

Requirements

- SCTP kernel module must be loaded (`lksctp-tools` package on Linux)
- All IP addresses must be routable from/to the peer
- Firewall rules must allow SCTP traffic on all configured IPs
- Both endpoints should be configured for multihoming for full redundancy

Limitations

- TCP transport does not support multihoming (only uses primary IP)
- TLS over SCTP multihoming may have compatibility limitations
- Path failover timing depends on kernel SCTP parameters

Reference Tables

Common 3GPP Application IDs

Application-Id	Interface	Description
16777251	S6a/S6d	MME/SGSN to HSS authentication and subscription data
16777252	S13/S13'	MME to EIR equipment identity check
16777238	Gx	PCEF to PCRF policy and charging control
16777267	S9	Home PCRF to Visited PCRF roaming policy
16777272	Sy	PCRF to OCS session binding
16777216	Cx	I-CSCF/S-CSCF to HSS IMS registration
16777217	Sh	AS to HSS IMS user data
16777236	SLg	MME/SGSN to GMLC location services
16777291	SLh	GMLC to HSS location subscriber info
16777302	S6m	MTC-IWF to HSS/HLR for M2M devices
16777308	S6c	SMS-SC/IP-SM-GW to HSS SMS routing
16777343	S6t	SCEF to HSS monitoring events
16777334	Rx	AF to PCRF media authorization

Common AVP Codes

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

Common Command Codes

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

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

Advanced Routing Module

The Advanced Routing module provides flexible, rule-based message routing capabilities with support for complex matching conditions.

Important: This module evaluates **inbound Diameter request packets only** (not answer packets). Answer packets follow the established session routing back to the originating peer - see Answer Routing for details.

Configuration

Enable the module and define routing rules in your configuration:

```
dra_module_advanced_routing:
  enabled: True
  rules:
    - rule_name: <rule_identifier>
      match: <match_scope>
      filters: [<filter_list>]
      route:
        peers: [<peer_list>]
```

Parameters

Parameter	Description
`enabled`	Set to `True` to activate the module
`rule_name`	Unique identifier for the routing rule
`match`	How filters are combined: `:all` (AND logic - all filters must match), `:any` (OR logic - at least one filter must match), `:none` (NOR logic - no filters can match)
`filters`	List of filter conditions (see Available Filters)
`route`	Routing action (see Route Actions below)

Route Actions

The `route` parameter supports multiple actions:

Route to Peers

```
route:
  peers: [peer01.example.com, peer02.example.com]
```

Routes to specified peer hostnames. Peers must be:

- Defined in the DRA's Diameter peer configuration
- The exact hostname as configured (case-sensitive)
- Currently connected for routing to succeed (disconnected peers are skipped)

Route to Destination-Host AVP

```
route: :destination_host
```

Routes to the peer specified in the message's Destination-Host AVP (293). If Destination-Host AVP is missing, routing falls back to normal behavior.

Drop Traffic

```
route: :drop
```

Silently discards the message without sending any response. Use for:

- Traffic filtering and blackholing
- Blocking unwanted requests
- Rate limiting by dropping excess traffic

Behavior:

- Message is dropped at DRA (not forwarded)
- No answer message is sent to requesting peer
- Implements Erlang Diameter `:discard` behavior
- Metric: `diameter_advanced_routing_drop_count_total` (see Prometheus Metrics)

Generate Error Response

```
route: {:error, 3004}
```

Generates a Diameter error answer with the specified Result-Code and sends it back to the requesting peer. Common result codes:

- `3002` - DIAMETER_UNABLE_TO_DELIVER (routing unavailable)
- `3003` - DIAMETER_REALM_NOT_SERVED (realm not supported)
- `3004` - DIAMETER_TOO_BUSY (overload protection, rate limiting)
- `5012` - DIAMETER_UNABLE_TO_COMPLY (general rejection)

Behavior:

- DRA generates error answer with specified Result-Code
- Answer includes Origin-Host, Origin-Realm, Session-Id (auto-populated by Diameter)
- Message is NOT forwarded to any peer
- Implements Erlang Diameter `{:protocol_error, code}` (equivalent to `{:answer_message, code}`)
- Metric: `diameter_advanced_routing_error_count_total` (see Prometheus Metrics)

Available Filters

Standard Filters

Available in both Advanced Routing and Advanced Transform:

- **`:application_id`** - Match Diameter application ID (see Application ID reference)
  - Single value: `{:application_id, 16777251}` (S6a/S6d)
  - Multiple values: `{:application_id, [16777251, 16777252]}` (S6a or S6b)
- **`:command_code`** - Match Diameter command code
  - Single value: `{:command_code, 318}` (AIR request)

    - Multiple values: `{:command_code, [317, 318]}` (ULR or AIR)

- **`:avp`** - Match AVP value (see AVP code reference)
    - Exact match: `{:avp, {296, "epc.mnc001.mcc001.3gppnetwork.org"}}`
    - Regex match: `{:avp, {1, ~r"999001.*"}}`
    - Multiple patterns: `{:avp, {1, ["505057001313606", ~r"999001.*", ~r"505057.*"]}}`
    - Any value (presence check): `{:avp, {264, :any}}`

Routing-Specific Filter

Only available in Advanced Routing:

- **`:via_peer`** - Match the peer where the request was received from
    - Single peer: `{:via_peer, "omnitouch-lab-dra01.epc.mnc001.mcc001.3gppnetwork.org"}`
    - Multiple peers: `{:via_peer, ["omnitouch-lab-dra01.epc.mnc001.mcc001.3gppnetwork.org", "omnitouch-lab-dra02.epc.mnc001.mcc001.3gppnetwork.org"]}`
    - Any peer: `{:via_peer, :any}`

Transform-Specific Filters

Only available in Advanced Transform:

- **`:to_peer`** - Match on predetermined destination peer (request packets only)
    - Single peer: `{:to_peer, "dra01.omnitouch.com.au"}`
    - Multiple peers: `{:to_peer, ["dra01.omnitouch.com.au", "dra02.omnitouch.com.au"]}`
- **`:from_peer`** - Match peer who sent the answer (answer packets only)
    - Single peer: `{:from_peer, "hss-01.example.com"}`
    - Multiple peers: `{:from_peer, ["hss-01.example.com", "hss-02.example.com"]}`
- **`:packet_type`** - Match packet direction

- Request: `{:packet_type, :request}`
- Answer: `{:packet_type, :answer}`

Important Filter Notes

- **AVP Filters**: Recommended for simple AVPs only (User-Name, Origin-Host, Destination-Realm, etc.)
  - Grouped AVPs are **not supported** and will not match
  - Complex binary values are **not supported**
  - Use format: `{:avp, {code, value}}`
- **List Operators**: Supported for all filter values except `:packet_type`
  - When a list is used, it applies **OR logic** within the list
  - Example: `{:command_code, [317, 318]}` matches command code 317 **OR** 318
- **Special Values**:
  - `:any` - Matches any value (checks for AVP presence)
  - Example: `{:avp, {264, :any}}` matches if Origin-Host AVP exists with any value

Routing Examples

Example 1: Via Peer Routing

Route messages based on which DRA they arrived from:

```
dra_module_advanced_routing:
  enabled: True
  rules:
    - rule_name: temporary_until_cutover_s6a_via_to_local_hss
      match: ":all"
      filters:
        - '{:application_id, 16777251}'
        - '{:via_peer, ["omnitouch-lab-
dra01.epc.mnc001.mcc001.3gppnetwork.org", "omnitouch-lab-
dra02.epc.mnc001.mcc001.3gppnetwork.org"]}'
        - '{:avp, {296, "epc.mnc001.mcc001.3gppnetwork.org"}}'
      route:
        peers: [omnitouch-lab-
hss01.epc.mnc001.mcc001.3gppnetwork.org, omnitouch-lab-
hss02.epc.mnc001.mcc001.3gppnetwork.org]
```

How it works: Routes S6a traffic that arrives via specific DRA peers to local HSS nodes.

Example 2: Inbound Roaming with Pattern Matching

Route roaming traffic based on IMSI patterns:

```
dra_module_advanced_routing:
  enabled: True
  rules:
    - rule_name: inbound_s6a_roaming_to_dcc
      match: ":all"
      filters:
        - '{:application_id, 16777251}'
        - '{:avp, {296, "epc.mnc001.mcc001.3gppnetwork.org"}}'
        - '{:avp, {1, ["505571234567", ~r"999001.*"]}}'
      route:
        peers: [dra01.omnitouch.com.au, dra02.omnitouch.com.au]
```

How it works: Routes S6a messages from specific Origin-Realm with matching IMSI patterns to designated DRA peers.

Example 3: Dynamic Routing with :destination_host

Route to the Destination-Host AVP value in the message:

```
dra_module_advanced_routing:
  enabled: True
  rules:
    - rule_name: route_to_specified_destination_host
      match: ":all"
      filters:
        - '{:avp, {1, [~r"90199.*"]}}'  # Match IMSI pattern
      route: :destination_host
```

How it works:

- When filters match, routes to the peer specified in the Destination-Host AVP (293)
- If Destination-Host AVP is missing, the match is considered a failure and falls back to normal routing
- Useful for honor routing when the sender specifies the exact destination

Example 4: Drop Unwanted Traffic

Drop traffic from specific IMSI ranges:

```
dra_module_advanced_routing:
  enabled: True
  rules:
    - rule_name: drop_test_subscribers
      match: ":all"
      filters:
        - '{:application_id, 16777251}'  # S6a
        - '{:avp, {1, [~r"999999.*"]}}'  # Test IMSI range
      route: :drop
```

How it works:

- Matches S6a messages with IMSI starting with 999999
- Silently drops the message without sending any response
- Useful for filtering test traffic or blocking specific subscriber ranges

- See Prometheus Metrics for monitoring dropped traffic

Example 5: Rate Limiting with Error Responses

Return DIAMETER_TOO_BUSY for specific traffic patterns:

```
dra_module_advanced_routing:
  enabled: True
  rules:
    - rule_name: rate_limit_high_volume_peer
      match: ":all"
      filters:
        - '{:via_peer, "mme-overloaded-01.example.com"}'
        - '{:application_id, 16777251}'
      route: {:error, 3004}
```

How it works:

- Matches S6a traffic from specific overloaded peer
- Returns DIAMETER_TOO_BUSY (3004) error response
- Requesting peer receives error and should back off
- Useful for overload protection and rate limiting
- See Prometheus Metrics for monitoring error responses

Example 6: Conditional Error Responses by Command

Block specific command types with appropriate error codes:

```
dra_module_advanced_routing:
  enabled: True
  rules:
    - rule_name: block_purge_requests
      match: ":all"
      filters:
        - '{:application_id, 16777251}'  # S6a
        - '{:command_code, 321}'         # PUR (Purge-UE-Request)
      route: {:error, 5012}
```

How it works:

- Matches S6a Purge-UE-Request messages
- Returns DIAMETER_UNABLE_TO_COMPLY (5012) error
- Blocks specific operations without dropping traffic silently
- Useful for selectively disabling certain Diameter commands

Advanced Transform Module

The Advanced Transform module enables dynamic modification of Diameter message AVPs based on matching criteria. See Rule Processing for details on how rules are evaluated.

Configuration

Enable the module and define transformation rules:

```
dra_module_advanced_transform:
  enabled: True
  rules:
    - rule_name: <rule_identifier>
      match: <match_scope>
      filters: [<filter_list>]
      transform:
        action: <transform_action>
        avps: [<avp_modifications>]
```

Parameters

Parameter	Description
`enabled`	Set to `True` to activate the module
`rule_name`	Unique identifier for the transform rule
`match`	How filters are combined: `:all` (AND logic), `:any` (OR logic), `:none` (NOR logic) - see Filter Logic
`filters`	List of filter conditions (see Available Filters)
`transform.action`	Type of transformation (`:edit`, `:remove`, or `:overwrite`)
`transform.avps`	List of AVP modifications to apply (see AVP code reference)

Transform Actions

Request Packets (Diameter Requests)

- **`:edit`** - Modify existing AVP values
  - Only modifies AVPs that exist in the message
  - If the AVP doesn't exist, no change is made
- **`:remove`** - Remove AVPs from the message
- **`:overwrite`** - Replace entire AVP structures
  - Requires `dictionary` parameter specifying the Diameter dictionary (e.g., `:diameter_gen_3gpp_s6a`)

Answer Packets (Diameter Answers)

- **`:remove`** - Remove AVPs from the message
- **`:overwrite`** - Replace entire AVP structures
  - Requires `dictionary` parameter

Important: If no rules match, the packet is passed through transparently without any transformations.

AVP Modification Syntax

Standard modification:

- `{:avp, {<code>, <new_value>}}` - Set AVP to new value

Removing AVPs:

- `{:avp, {<code>, :any}}` - Remove AVP by ID (removes regardless of current value)
- Note: Removing based on avp_id is supported; removing based on AVP contents is not supported

Overwrite with dictionary:

```
transform: %{
  action: :overwrite,
  dictionary: :diameter_gen_3gpp_s6a,
  avps: [{:avp, {:"s6a_Supported-Features", {:"s6a_Supported-
Features", 10415, 1, 3221225470, []}}}]
}
```

Transform Examples

Example 1: To-Peer Based Realm Rewriting

Rewrite Destination-Realm based on where the message is being routed:

```
dra_module_advanced_transform:
  enabled: True
  rules:
    - rule_name: rewrite_s6a_destination_realm_for_Operator_X
      match: ":all"
      filters:
        - '{:to_peer, ["dra01.omnitouch.com.au",
"dra02.omnitouch.com.au"]}'
        - '{:avp, {296, "epc.mnc001.mcc001.3gppnetwork.org"}}'
        - '{:avp, {1, [~r"9999999.*"]}}'
      transform:
        action: ":edit"
        avps:
          - '{:avp, {283, "epc.mnc999.mcc999.3gppnetwork.org"}}'
```

How it works: When S6a requests are routed to specific DRA peers and match the IMSI pattern, rewrites the Destination-Realm for Operator X network.

Example 2: Multiple Carrier Routing with Transforms

```
dra_module_advanced_transform:
  enabled: True
  rules:
    - rule_name:
rewrite_s6a_destination_realm_for_roaming_partner_ausie
      match: ":all"
      filters:
        - '{:to_peer, ["dra01.omnitouch.com.au",
"dra02.omnitouch.com.au"]}'
        - '{: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: Routes different IMSI subscriber ranges to appropriate network realms based on IMSI patterns. First matching rule wins (see Execution Order).

Example 3: MVNO Realm Rewriting

```
dra_module_advanced_transform:
  enabled: True
  rules:
    - rule_name: rewrite_s6a_destination_realm_for_single_sub
      match: ":all"
      filters:
        - '{:to_peer, ["dra01.omnitouch.com.au",
"dra02.omnitouch.com.au"]}'
        - '{:avp, {296, "epc.mnc001.mcc001.3gppnetwork.org"}}'
        - '{:avp, {1, ["505057000003606"]}}'  # Exact IMSI match
      transform:
        action: ":edit"
        avps:
          - '{:avp, {283, "epc.mnc001.mcc001.3gppnetwork.org"}}'
```

How it works: Transforms Destination-Realm for specific MVNO subscriber to their hosted core network.

Example 4: Request-Only Transform with Packet Type Filter

Transform only request packets (not answers):

```
dra_module_advanced_transform:
  enabled: True
  rules:
    - rule_name: Tutorial_Rule_AIR
      match: ":all"
      filters:
        - '{:application_id, 16777251}'
        - '{:command_code, 318}'
        - '{:packet_type, :request}'
        - '{:avp, {1, "999999000000001"}}'
        - '{:avp, {264, :any}}'  # Origin-Host must exist with any
value
      transform:
        action: ":edit"
        avps:
          - '{:avp, {1, "999999000000002"}}'
```

How it works:

- Matches only S6a AIR **request** packets (not answer packets)
- Checks User-Name (AVP 1) equals "999999000000001"
- Verifies Origin-Host (AVP 264) exists with any value
- Rewrites User-Name to "999999000000002"
- If AVP doesn't exist, no change is made

Example 5: Remove AVP

Remove specific AVP from messages:

```
dra_module_advanced_transform:
  enabled: True
  rules:
    - rule_name: remove_user_name_avp
      match: ":all"
      filters:
        - '{:application_id, 16777251}'
      transform:
        action: ":remove"
        avps:
          - '{:avp, {1, :any}}'  # Remove User-Name regardless of
value
```

How it works: Removes User-Name AVP (code 1) from all S6a messages, regardless of its current value.

Example 6: Overwrite Grouped AVP on Answer Packets

Modify complex grouped AVPs in answer packets using the `:overwrite` action with dictionary support:

```
dra_module_advanced_transform:
  enabled: True
  rules:
    - rule_name: add_sos_apn_to_ula
      match: ":all"
      filters:
        - '{:application_id, 16777251}'       # S6a/S6d
        - '{:command_code, 316}'              # ULA (Update
Location Answer)
        - '{:packet_type, :answer}'           # Answer packets
only
        - '{:avp, {296, "epc.mnc001.mcc001.3gppnetwork.org"}}'  #
Origin-Realm
      transform:
        action: ":overwrite"
        dictionary: ":diameter_gen_3gpp_s6a"
        avps:
          - '{:avp, {:"s6a_APN-Configuration-Profile",
              {:"s6a_APN-Configuration-Profile", 1, 0, [
                {:"s6a_APN-Configuration", 1, 0, "internet", [],
                  [{:"s6a_EPS-Subscribed-QoS-Profile", 9,
                    {:"s6a_Allocation-Retention-Priority", 1, [0],
[0], []}, []}],
                  [1], [], [], [1], ["0800"],
                  [{:s6a_AMBR, 4200000000, 4200000000, [], [],
[]}],
                  [], [], [], [], [], [], [], [], [], [], [], [],
[], [], []},
                {:"s6a_APN-Configuration", 2, 0, "ims", [],
                  [{:"s6a_EPS-Subscribed-QoS-Profile", 5,
                    {:"s6a_Allocation-Retention-Priority", 1, [0],
[1], []}, []}],
                  [0], [], [], [1], ["0800"],
                  [{:s6a_AMBR, 4200000000, 4200000000, [], [],
[]}],
                  [], [], [], [], [], [], [], [], [], [], [], [],
[], [], []},
                {:"s6a_APN-Configuration", 3, 0, "sos", [],
                  [{:"s6a_EPS-Subscribed-QoS-Profile", 5,
                    {:"s6a_Allocation-Retention-Priority", 1, [0],
[1], []}, []}],
                  [1], [], [], [1], ["0800"],
                  [{:s6a_AMBR, 4200000000, 4200000000, [], [],
```

```
[]}],
                  [], [], [], [], [], [], [], [], [], [], [], [],
[], [], []}
              ], []}
            }}'
```

How it works:

- Matches S6a Update Location Answer (ULA) packets from a specific Origin-Realm
- Uses `:overwrite` action to replace the entire APN-Configuration-Profile grouped AVP
- **Requires `dictionary` parameter** to properly encode complex grouped AVP structures
- Adds three APN configurations: "internet" (context 1), "ims" (context 2), and "sos" (context 3)
- Each APN includes QoS profiles, bandwidth limits (AMBR), and PDN type settings
- The transformation ensures emergency services (SOS) APN is provisioned for all subscribers from this realm

When to use `:overwrite` with dictionary:

- Modifying grouped AVPs with nested structures (like APN-Configuration-Profile)
- Adding or restructuring complex 3GPP subscription data
- When `:edit` action cannot handle the AVP complexity
- Dictionary must match the Diameter application (`:diameter_gen_3gpp_s6a` for S6a, etc.)

Important notes:

- `:overwrite` replaces the entire AVP, not just individual fields
- The AVP structure must match the dictionary definition exactly
- Incorrect structure will cause encoding failures and dropped packets
- This is an advanced feature - validate thoroughly in test environment first

Use Cases

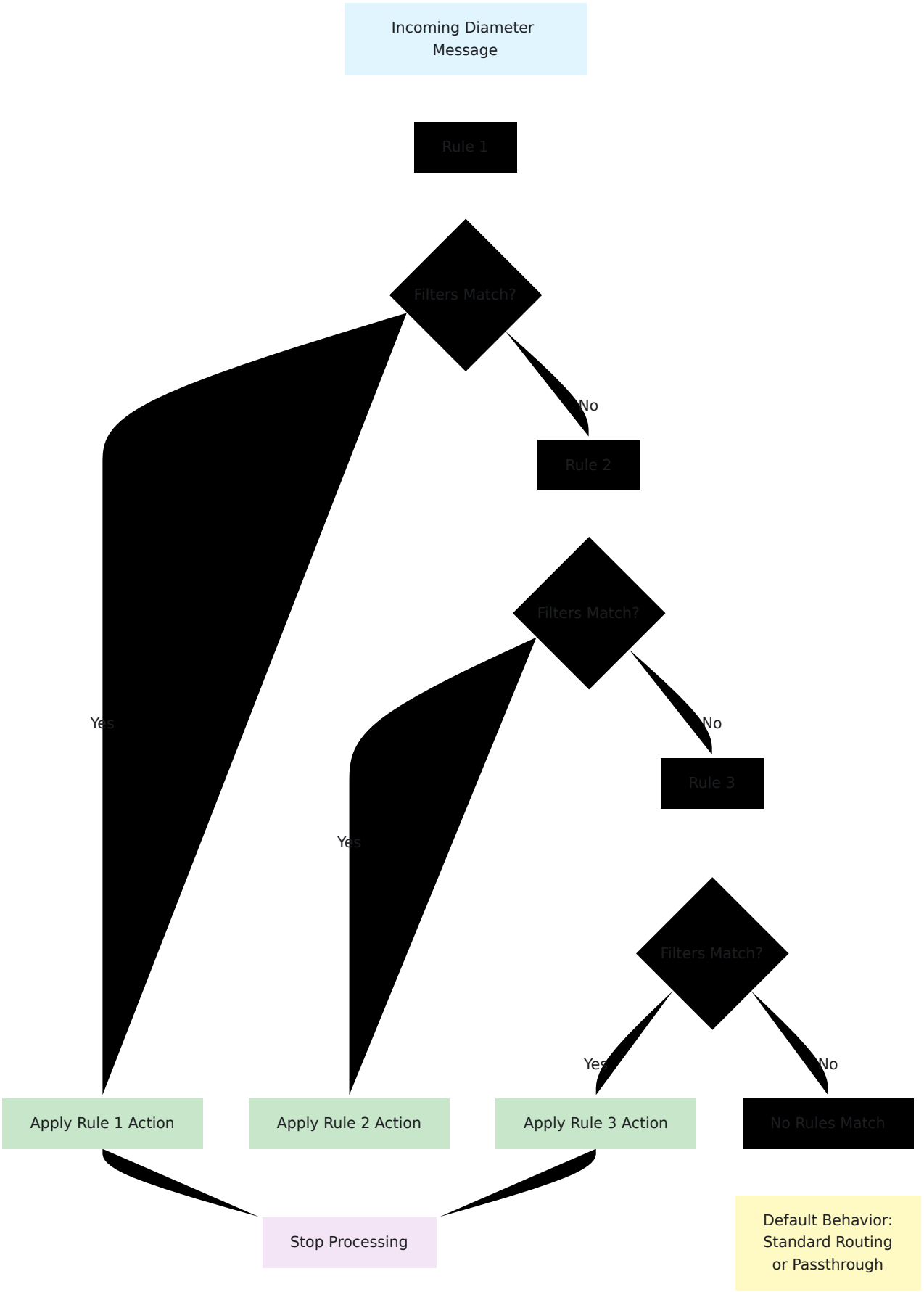
- **MVNO Support**: Route virtual operator traffic to hosted core networks
- **Network Migration**: Gradually redirect subscribers to new infrastructure
- **Realm Translation**: Convert between different naming schemes for roaming partners
- **Multi-tenancy**: Isolate subscriber populations by realm
- **Carrier Routing**: Direct traffic to correct carrier networks based on IMSI ranges

---

Rule Processing

Applies to both Advanced Routing and Advanced Transform modules.

Execution Order



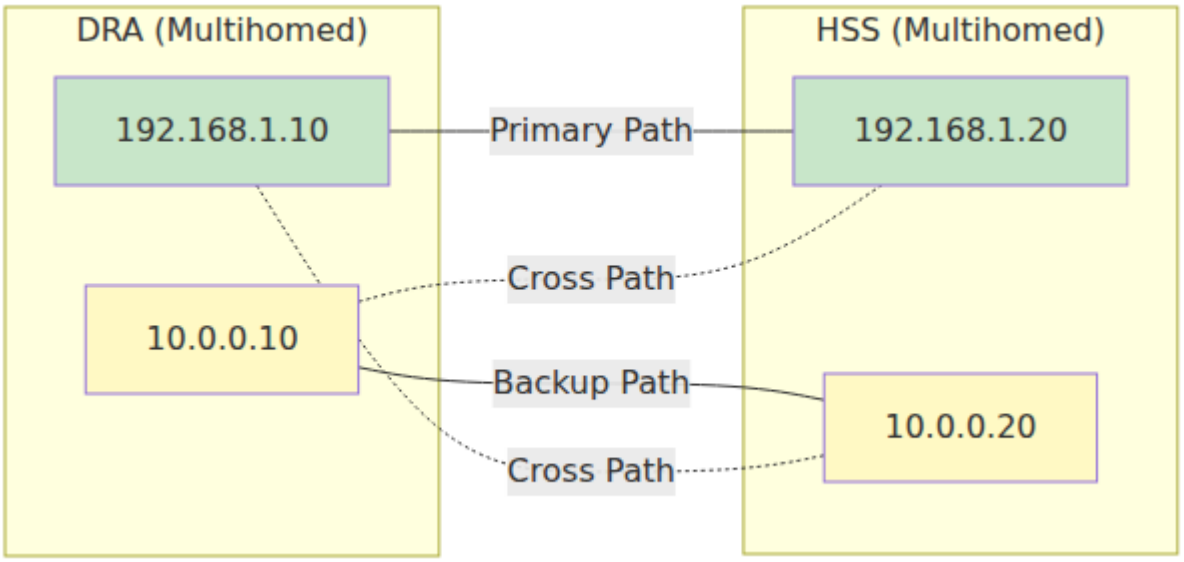
1. Rules are evaluated **in order from top to bottom** as defined in configuration
2. Filters within a rule are evaluated based on the `match` parameter (`:all`, `:any`, or `:none`)
3. **First matching rule wins** - subsequent rules are not evaluated
4. If no rules match, default routing/passthrough behavior is used

Filter Logic

The `match` parameter determines how filters are combined:

match: :all (AND Logic)

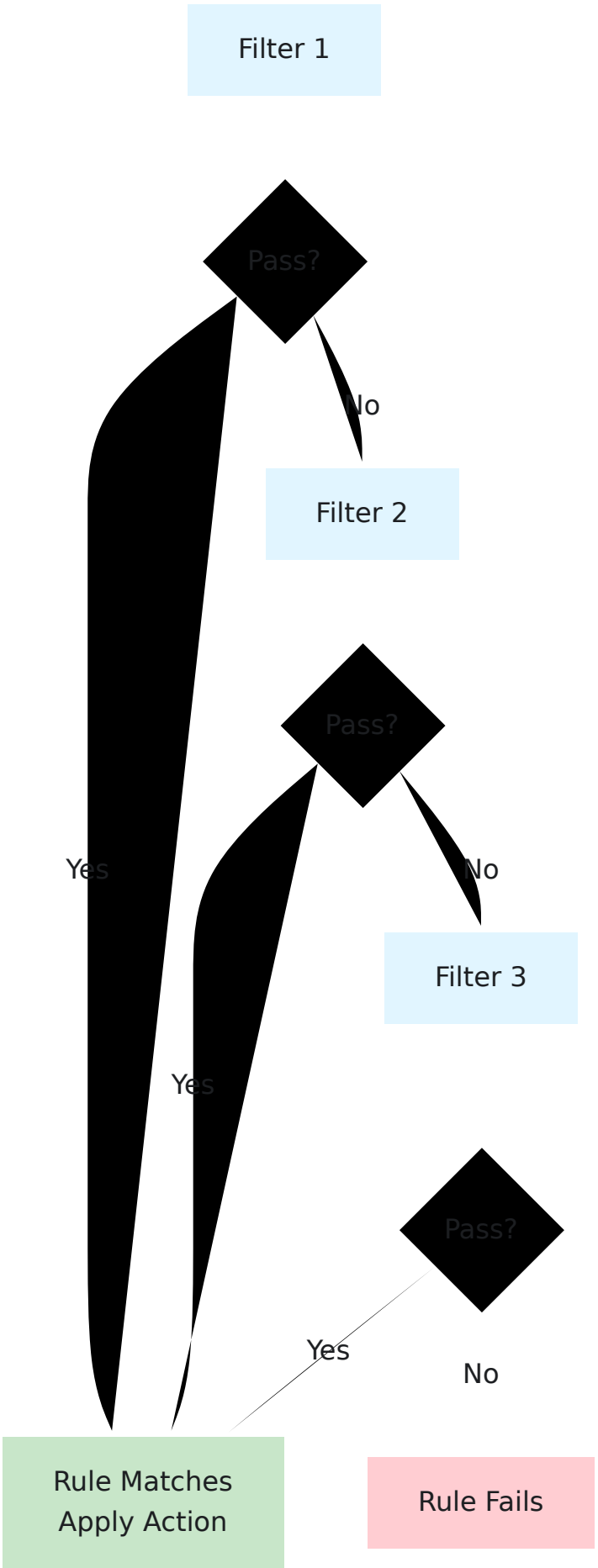
All filters must match for the rule to succeed.



Example: With 3 filters, `filter1 AND filter2 AND filter3` must all be true.

match: :any (OR Logic)

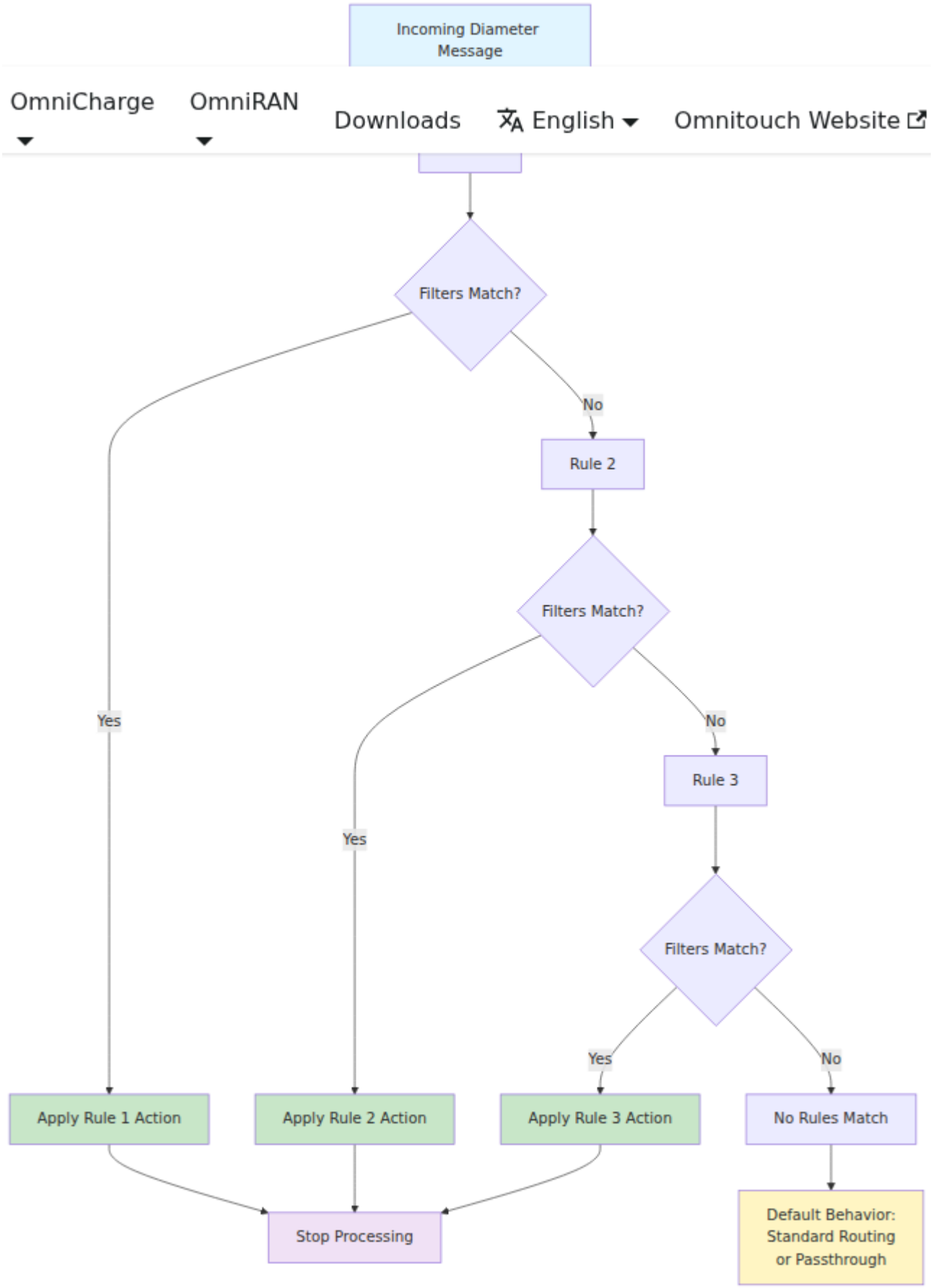
At least one filter must match for the rule to succeed.


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

Example: With 3 filters, `filter1 OR filter2 OR filter3` (any one passes).

match: :none (NOR Logic)

No filters can match for the rule to succeed (inverse matching).



Example: With 3 filters, `NOT filter1 AND NOT filter2 AND NOT filter3` (all must fail).

Additional Notes:

When using list operators within a filter value (e.g., `{:avp, {1, ["value1", "value2"]}}`), the values use **OR** logic (any can match).

Regular Expression Patterns

Use `~r"pattern"` syntax for regex matching:

- `~r"999001.*"` - Matches IMSI starting with 999001
- `~r"^310[0-9]{3}.*"` - Matches IMSI with specific MNC patterns
- `~r".*test$"` - Matches values ending with "test"

Best Practices

1. **Specificity**: Order rules from most specific to most general
2. **Performance**: Place most common matches first to reduce processing overhead
3. **Testing**: Validate regex patterns before deployment
4. **Documentation**: Use descriptive `rule_name` values for operational clarity
5. **Monitoring**: Track rule match rates to verify expected behavior

---

Extended Metrics Module

The Extended Metrics module provides advanced telemetry and analytics capabilities for analyzing Diameter traffic patterns beyond the standard metrics.

Configuration

Enable the module and configure specific metric types:

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

Parameters

Parameter	Description
`enabled`	Set to `true` to activate the extended metrics module
`attach_attempt_reporting_enabled`	Enable tracking and reporting of LTE attach attempts (S6a AIR/AIA)

Available Metrics

Attach Attempt Tracking

Tracks LTE subscriber attach attempts by monitoring Authentication Information Request (AIR) and Answer (AIA) message pairs:

Parse error on line 36: ... style Metrics fill:#f3e5f5 style E ----------^
Expecting 'SOLID_OPEN_ARROW', 'DOTTED_OPEN_ARROW', 'SOLID_ARROW', 'BIDIRECTIONAL_SOLID_ARROW', 'DOTTED_ARROW', 'BIDIRECTIONAL_DOTTED_ARROW', 'SOLID_CROSS', 'DOTTED_CROSS', 'SOLID_POINT', 'DOTTED_POINT', got 'TXT'

Try again

Measurement: `attach_attempt_count`

Fields:

- `imsi` - The subscriber IMSI (from User-Name AVP - see AVP codes)

Tags:

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

How it works:

1. When an AIR (command code 318, S6a application 16777251 - see Application IDs) is received, the module extracts:
   - End-to-End-ID for request/response correlation
   - IMSI (User-Name AVP code 1)
   - Origin-Host (AVP code 264)
2. Request metadata is stored in ETS with TTL
3. When the matching AIA is received, the module:
   - Correlates using End-to-End-ID
   - Extracts the result code (AVP 268 or experimental result code AVP 297)
   - Emits the metric with IMSI, origin host, and result code

Use Cases

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

Integration

Extended metrics are exported via InfluxDB integration:

```
DRA.Metrics.InfluxDB.write(%{
  measurement: "attach_attempt_count",
  fields: %{imsi: "505057000000001"},
  tags: %{origin_host: "mme-01.example.com", result_code: 2001}
})
```

Result codes are standard Diameter codes:

- `2001` - Success (DIAMETER_SUCCESS)
- `5001` - Authentication failure (DIAMETER_AUTHENTICATION_REJECTED)
- `5004` - Diameter AVP unsupported
- See RFC 6733 for complete result code list

Important Notes

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

---

Prometheus Metrics

The DRA exposes comprehensive Prometheus metrics for monitoring Diameter traffic, peer health, and module operations. All metrics are available at the `/metrics` endpoint.

Core Diameter Metrics

Peer Status

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

- `origin_host` - Peer's Diameter Identity
- `ip` - Peer's IP address

Example:

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

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

Message Count

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

- `origin_host` - Peer's Diameter Identity
- `received_from` - Peer the message was received from
- `application_id` - Diameter Application-Id (see Application ID reference)
- `cmd_code` - Diameter Command-Code (see Common 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:

```
# 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]))
```

Response Result Codes

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

- `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:

```
# 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:

- `2001` - DIAMETER_SUCCESS
- `3002` - DIAMETER_UNABLE_TO_DELIVER
- `3003` - DIAMETER_REALM_NOT_SERVED
- `3004` - DIAMETER_TOO_BUSY
- `5001` - DIAMETER_AUTHENTICATION_REJECTED
- `5004` - DIAMETER_INVALID_AVP_VALUE
- `5012` - DIAMETER_UNABLE_TO_COMPLY

Response Delay

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

- `origin_host` - Original requester

- `routed_to` - Peer that sent the answer
- `application_name` - Application name
- `cmd_name` - Command name

Example:

```
# Average response time from HSS
avg(diameter_peer_last_response_delay{routed_to="hss01.example.com"})

# P95 response time for S6a
histogram_quantile(0.95,
  rate(diameter_peer_last_response_delay{application_name="3GPP_S6a"}
[5m])
)
```

Unanswered Requests

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

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

Example:

```
# 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 Connection Attempts

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

- `origin_host` - Unauthorized peer's claimed identity
- `supported_applications` - Applications advertised by peer
- `peer_ip` - IP address of connection attempt

Example:

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

# Alert on unauthorized access
diameter_peer_unauthorized_connection_count_total > 0
```

Advanced Routing Module Metrics

Dropped Traffic

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

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

Example:

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

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

When traffic is dropped:

- Advanced routing rule matches with `route: :drop`
- Message is silently discarded per Erlang Diameter `:discard` behavior
- No answer message is sent to the requesting peer
- See Advanced Routing Module for configuration

Error Responses

Metric: `diameter_advanced_routing_error_count_total` **Type**: Counter **Description**: Error responses generated by advanced routing `{:error, result_code}` action **Tags**:

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

Example:

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

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

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

When error responses are generated:

- Advanced routing rule matches with `route: {:error, result_code}`

- DRA generates Diameter answer with specified Result-Code
- Answer is sent back to requesting peer (not forwarded)
- Per Erlang Diameter: `{:protocol_error, code}` equivalent to `{:answer_message, code}`
- See Advanced Routing Module for configuration

Common Error Codes Used:

- `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)

Important Notes

- All metrics are available at the `/metrics` endpoint in Prometheus format
- **Extended Metrics Module** provides additional S6a-specific metrics (see Extended Metrics Module)
- **BEAM/Erlang metrics** are also exposed but not documented here
- All counters are cumulative and should be queried with `rate()` for per-second rates
- High cardinality tags (like IMSI) are only used in Extended Metrics module to avoid metric explosion

---

Troubleshooting

Rule Not Matching

- Verify all filter conditions are correct
- Check AVP codes match your Diameter application (see AVP code reference)
- Test regex patterns independently (see Regular Expression Patterns)
- Ensure message type matches `match` scope (see Filter Logic)

- Review Available Filters to ensure you're using the correct filter type for your module

Unexpected Routing

- Review rule ordering - first match wins
- Verify peer names are correct and reachable
- Check for conflicting rules with overlapping filters
- Confirm Standard Diameter Routing behavior when no rules match

Transform Not Applied

- Confirm AVP codes are correct for your use case (see AVP code reference)
- For `:edit` action: Verify the AVP exists in the message (edit won't create new AVPs)
- Check that filters match the intended message flow
- Verify packet type filter if used (`:request` vs `:answer`)
- Ensure action is supported for packet type (`:edit` only works on requests - see Transform Actions)
- Review Rule Processing for execution order