# BIG-IP® Service Provider: SIP Administration

Version 13.0

# Table of Contents

# Introduction to SIP Message Routing Framework

## Introduction to SIP Message Routing Framework

A BIG-IP® system provides advanced Message Routing Framework (MRF) capabilities. The BIG-IP SIP solution is based on MRF framework. This guide is designed to introduce the reader to BIG-IP MRF and SIP concepts. Finally, various SIP use cases will be documented to help the reader with their deployment needs.

# SIP Overview

## SIP Overview

A BIG-IP® system's message routing framework (MRF) SIP solution provides high scalability, availability, and reliability to SIP Proxies, Session Border Controllers, Media Servers and many other SIP devices. The BIG-IP LTM can distribute and balance SIP and RTP traffic among multiple SIP devices, to help maintain availability, under high call volumes. Additionally, the F5 solution can perform advanced health checks on the SIP devices, routing SIP clients away from unstable or unreliable devices and providing increased reliability to existing SIP solutions.

### Capabilities

This section provides a concise summary of the BIG-IP® MRF SIP solution.

#### Load Balancing

- Route SIP control messages, without modifying SIP headers.
- The following headers can be configured to be automatically modified.
  - VIA Header inserted
  - Record Router Header inserted
  - Decrementing max forwards
- Any header attribute can be modified via iRule.
- Natively route messages based on:
  - From URI
  - To URI
  - Request URI
  - Originating Virtual Server
- Route messages via iRule on any attribute of the message.
- Response routing natively using data added to the inserted VIA Header.
- Response routing available via iRule:
  - Add private header to request
  - Insertion of VIA Header to request via iRule
  - Route to upstream device using received VIA Header
  - Remember data from request processing
- Bi-directional persistence support.
  - Persistence key selection via configuration or custom key via iRule
- Connection Re-Use Support
- High Availability (HA)
  - Connection mirroring
  - Persistence table replication

#### ALG without SNAT (No Address Translation)

- Snoop control messages flowing through to manage media flows.
- iRule can be used to rewrite headers.
- Create media records in session db.

- Create deny listeners to drop media packets received before the callee responds with its media details.
- Create media flows to forward packets between caller and callee.
- High Availability (HA)
    - Call table replication (supports failback)
    - Control connection mirroring (can be recreated on failback by endpoint)
    - Media flow mirroring

### SRTP Compliance (RFC 3711)

We do not support SRTP in ALG without SNAT mode.

# Operation Modes

## Operation Modes

### Load Balancing

#### Configuration Objects

##### Virtual Server

A virtual server is a traffic-management object on the BIG-IP® system that is represented by an IP address and a service. Clients on an external network can send application traffic to a virtual server, which then directs the traffic according to your configuration instructions.

The virtual server configuration contains a destination address and mask which specifies what IP addresses and port the virtual server will listen for incoming packets. The virtual server object also contains a source address allowing it to limit packets to those packets that originate from a range of devices.

The behavior of a virtual server is determined by the set of profiles attached to the virtual server. To configure a MRF SIP virtual server, transport profile (TCP, UDP or SCTP), a sipsession profile and a siprouter profile must be attached to the virtual server.

The behavior of a virtual can further be extended by assigning iRules to the virtual.

All virtual servers that share the same siprouter profile will share the same router instance. The routing instance owns the route table, the persistence table and flowmap table (a table of all open connection usable for message forwarding).

- A virtual server of type "message-routing" assigned with "session" & "router" profile is part of a SIP router-instance.
- Persistence is configured through the SIP session profile, hence a persist profile must not be attached to the virtual server.
- One or more iRules can be attached. The rules are validated against the configured transport and the events/commands as supported by SIP and MRF.
- Source-address-translation can be configured in the virtual server. For server side flows the transport-config parameters overrides the virtual server setting for source address translation. If a route does not specify a transport-config, than the transport of the originator of the message will be as the transport for the outgoing connection. For example if the SIP message originated on a virtual server, the parameters of the virtual server will be used to create the outgoing connection. This includes the source-address-translation settings of the virtual server.
- A static route object containing wildcard attributes can be used as a default route. The configured by adding a SIP route with empty request-uri, from-uri and to-uri attributes to the siprouter profile used by the virtual server.
- Static routes can be configured to only operate on messages originating for specific virtuals. This allows each virtual using the same router instance to have independent default routes.
- The virtual can be configured to listen on all or selected set of VLANs.
- All virtuals that needs to work together as one router-instance must share the same "router" profile.
- All virtuals that are to acts as one router-instance, must have the all their corresponding Virtual Addresses in the same traffic group. The traffic group of the siprouter profile must match the traffic-group of the virtual's virtual address.

```
ltm virtual <virtual name> {
    cmp-enabled yes
    destination <ip address>:<port>
    enabled | disabled
    ip-protocol <tcp|udp|sctp>
    mask <netmask>
    profiles {
        siprouter
        sipsession
        <tcp|udp|sctp>
        <other profiles>
    }
    rules <list of iRules|none>
    source 0.0.0.0/0
    vlans {
        <list of vlans>
    }
    vlans-enabled | vlans-disabled
}
```

### SIP Router Profile

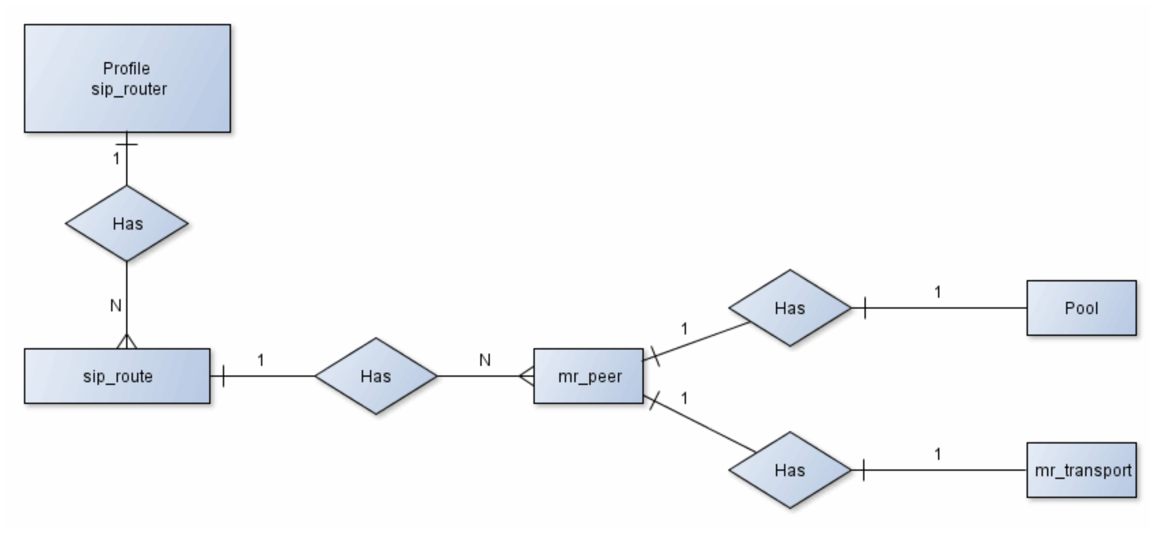


**Figure 1: SIP Router Profile Entity-Relationship**

- A SIP router profile provides the router-instance level characteristics such as mode of operation, routes and more. This profile defines attributes that are to be the same across the entire router-instance. In addition, it holds the static routes to be used across the entire router-instance.
- A route is made of peers, where a peer may have a pool and a transport config
- If a peer does not contain a pool, the local address of the originating connection is used as the remote address of the outgoing connection
- If a peer does not contain a transport-config, the settings of the originating connections transport (virtual or transport-config) is used to create the outgoing connection.
- The router profile binds the multiple virtuals (that clients connect to) and peers (that connect to backend servers), together with common shared states.
- For ALG, no routes are configured for the router profile. The local address of the originating flow is used as the remote address of the outgoing connection.

```
ltm message-routing sip profile router siprouter {
    defaults-from none
    inherited-traffic-group <true|false>
    max-pending-bytes <integer>
    max-pending-messages <integer>
    max-retries <integer>
    mirror <enabled|disabled>
    mirrored-message-sweeper-interval <integer>
    operation-mode load-balancing
    routes {
        <static routes>*
    }
    session {
        max-session-timeout <integer>
        transaction-timeout <integer>
    }
    traffic-group <traffic group name>
    use-local-connection <enabled|disabled>
}
```

**Table 1: SIP router profile attributes**

| Attribute | Description | Type | Acceptable Values | Default |
|---|---|---|---|---|
| operation-mode | Sets the operation mode of the SIP routing instance.<br><br>Must be set to load-balancing | ENUM | load-balancing<br><br>application-level-gateway | load-balancing |
| routes | List of references to sip-route object. The ordering of the route entries does not matter. | sip-route | A list of sip-route objects | |
| max-pending-messages | The maximum number of pending messages that are held while waiting for a connection to a peer to be created. Once reached any additional messages to the peer is flagged as undeliverable and returned to the originator. | UINT32 | 1 to 4294967295 (32-bits) | 23768 |
| max-pending-bytes | The maximum number of bytes contained within pending messages that is held while waiting for a connection to a peer to be created. Once reached any additional messages to the peer is flagged as undeliverable and returned to the originator. | UINT32 | 1 to 4294967295 (32-bits) | 64 |
| use-local-connection | Controls whether connections established by the ingress TMM are preferred over connections established by other TMMs when selecting egress connection to destination peer. | BOOL | Enabled/ Disabled | Enabled |
| traffic-group | The traffic group for the router instance. All virtual servers using this router profile will have the their traffic group replaced by the traffic group of the router profile. | traffic-group | | default (inherited from partition) |
| mirror | Enables mirroring of all incoming connections for all virtual servers using this router instance, and all outgoing connection created by this router instance. | BOOL | enabled/ disabled | disabled |

| Attribute | Description | Type | Acceptable Values | Default |
|---|---|---|---|---|
| mirrored-message-sweeper-interval | This attribute sets the frequency of the mirrored message sweeper. For virtual servers where mirroring is enabled, the received messages will be processed on both the active device and the standby device. On the standby device, the messages are not routed, instead they are stored in a message store until the active device sends a notification that the message has been routed to the standby device so that the standby device can deliver the message to the equivalent connection for egress processing. A sweeper has been implemented to drop messages from the message store if they remain in the store longer than the time specified in this attribute. The time shall be in milliseconds. | UINT32 | | 1000(ms) |
| media-proxy | | | | |
| max-media-sessions | N/A in LB Mode | UINT32 | 1 to 10 (32-bits) | 6 |
| media-inactivity-timeout | N/A in LB Mode | UINT32 | 1 to 120 (32-bits) | 120 |
| session | | | | |
| dialog-establishment-timeout | N/A in LB Mode | UINT32 | 1 to 4294967295 (32-bits) | 32 |
| max-session-timeout | N/A in LB Mode | UINT32 | 1 to 4294967295 (32-bits) | 7200 |
| transaction-timeout | Specifies the maximum time in seconds between request and its response. A provisional response restarts the timer. This may not affect all transactions. The scenarios where a BIG-IP® system waits for response (like final response for REGISTER request), is impacted, by dropping any persistent data maintained for this request. | UINT32 | 1 to 4294967295 (32-bits) | 180 |

### *Operation Mode*

**Table 2: SIP operation mode**

| operation-mode | Description |
|---|---|
| load-balancing | Configures the SIP routing instance to operate in load-balancing mode. See *How to configure DAG Modes* for details. |
| application-level-gateway | Configures the SIP routing instance to operate in application level gateway mode (ALG). See *Default DAG* for details |

### *SIP Route Table*

- SIP routes are collected into a route table.
- Each SIP router instance maintains a route table.
- When the SIP router receives a message for forwarding, the route table is used to determining the best route to use for forwarding the message.
- The message's to-uri (RFC 3261 section 8.1.1.2), from-uri (RFC 3261 section 8.1.1.3), request-uri (RFC 3261 section 8.1.1.1) attributes and originating virtual is matched against the routes in the route table.

**Table 3: SIP Route table example**

| Request-URI | From-URI | To-URI | Virtual | Route Value |
|---|---|---|---|---|
| | | | | Default_pool |
| | | *.f5.com | | Subdomain pool |
| | | *@f5.com | | F5 domain pool |
| | | help@f5.com | | Helpdesk pool |
| | *@external.com | | | External pool |
| | priority.user@isp.net | help@f5.com | | Special helpdesk pool |

## Attributes Matching

- The attributes is matched in the following order: to-uri, from-uri, request-uri and virtual.
- Each URI is matched starting at the end of the attribute.
- Because a URI key may contain a wildcard, a URI from a messages attribute may match multiple attributes, the longest match for the attribute is tried first.
- If a matching route does not exist using the longest match, the next longest match is attempted.
- An empty field is the same as a wildcard (all values are considered to match).
- Route selection first matches the to-uri then the from-uri, followed by the request-uri and finally the virtual.
- Each field is matched starting at the end of the field (last character).
- There can be no characters before a wildcard (asterisk).

## Specific Route Match Example

Consider a SIP route table with following routes:

**Table 4: SIP route match**

| To-URI | From-URI | Request-URI | Virtual | Route Value |
|---|---|---|---|---|
| | | | | default-route |
| *.f5.com | | | | subdomain-route |
| *af5.com | | | | f5domain-route |
| help@f5.com | | | | helpdesk-route |
| it@f5.com | | | internal_vs | it-route |

To route a message with the following attributes:

**Table 5: SIP route match**

| To-URI | From-URI | Request-URI | Virtual |
|---|---|---|---|
| it@f5.com | top.salesman@vendor.com | it@f5.com | external_vs |

- The SIP route table (refer table 3.3.3) would first look for a match for the to-uri.
- It would find 3 matches: "" (wildcard), "*@f5.com", and "it@f5.com".
- The longest match would be "it@f5.com". It would then try to match from-uri, request-uri and virtual.
- The from-uri attribute would match the wildcard as would the request-uri.
- The virtual would not match.
- No match was found using "it@f5.com", so it would return to the next longest matching value, "*@f5.com".
- It would then try and match from-uri, request-uri and virtual.
- Matches for all three fields would be found so it would forward the message to a host as specified in the route value of the f5 domain route in table 3.3.3

### SIP Route

```
ltm message-routing sip route siproute {
    from-uri <string>
    peer-selection-mode <sequential|ratio>
    peers {
        <one or more peer>
    }
    request-uri <string>
    to-uri <string>
    virtual-server <string>
}
```

The SIP route has the following attributes.

**Table 6: SIP Route Attributes**

| Attribute | Description | Type | Acceptable Values | Default |
|---|---|---|---|---|
| name | Specifies the name of the route object | STRING | ASCII string | None |
| request-uri | Defines the pattern to be matched against the request-uri of a sip message. This URI is matched as a case insignificant method. It should be in the form of *user@domain.* The sip: prefix should not be present. Any additional modifiers (for example port or | STRING | ASCII string<br><br>format: <user@domain> | "" |

| Attribute | Description | Type | Acceptable Values | Default |
|---|---|---|---|---|
| | transport) should also not be present. It may begin with a wildcard, '*'. If empty, it is treated as if the entire URI was a wildcard (matching all Request-URIs). | | | |
| to-uri | Defines the pattern to be matched against the To field of a sip message. This URI is matched as a case insignificant method. It should be in the form of *user@domain.* The sip: prefix should not be present. Any additional modifiers (for example port or transport) should also not be present. It may begin with a wildcard, '*'. If empty, it is treated as if the entire URI was a wildcard (matching all To-URIs). | STRING | ASCII string<br>format: <user@domain> | "" |
| from-uri | Defines the pattern to be matched against the From field of a sip message. This URI is matched as a case insignificant method. It should be in the form of *user@domain.* The sip: prefix should not be present. Any additional modifiers (for example port or transport) should also not be present. It may begin with a wildcard, '*'. If empty, it is treated as if the entire URI was a wildcard (matching all From-URIs). | STRING | ASCII string<br>format: <user@domain> | "" |
| virtual-server | Specifies a virtual server that this route is limited to. If no virtual is specified, messages originating on any connection may be routed to the route. | virtual-server | A virtual server instance | None |
| peers | Specifies the list of peers. The peers attribute is a list of references to mr-peer objects. | mr-peer | An instance or mr-peer | |
| peer-selection-mode | Describes the method of selecting a peer from a list of peers.<br>**sequential**: Peers are selected in the order listed. All traffic is routed to the first peer unless all pool members in the peer are marked down.<br>**ratio:** Peers are selected based on their weights in comparison with other peers. | ENUM | sequential/ratio | sequential |

A SIP route specifies a set of peers to use for forwarding messages. Each route contains a route key and a route value. The route key contains attributes that are matched against attributes in a SIP message. The route value contains a list of peers. If the attributes in the route key match, the message is forwarded to a host specified by route value.

### *Route Key*

The route key contains the attributes that are matched against attributes from the SIP message header and optionally a list of virtual servers.

- The to-uri, from-uri and request-uri attributes are matched against corresponding attributes in a SIP message's header.

- These values are matched in a case insignificant method.
- Only the user@host portion of the uri is matched. The protocol prefix and additional modifiers (like port, transport, key, etc) are not included in the match. (see RFC 3261 section 19.1)
- The uri key in the message may start with a wildcard character, '*' (for example '*@f5.com'). If a uri key starts with a wildcard, this means that any valid pattern of characters at that position in the message's corresponding attribute is considered as matching refer section 3.3.1.1
- An empty uri key is considered as matching any valid value in the message's corresponding attribute.
- If virtual server attribute in the route key is empty, the route is applied to all messages. If the virtual server attribute is not empty, the route applies only to messages originating the virtual server specified.
- A route key with all fields empty (wildcard) is used as a default route.

**Table 7: Filtered URI for Route Key**

| Example URI | Filtered URI for matching |
|---|---|
| sip: alice@atlanta.com | alice@atlanta.com |
| sip: alice:secretword@atlanta.com;transport=tcp | alice@atlanta.com |
| sips:alice@atlanta.com?subject=project %20x&priority=urgent | alice@atlanta.com |
| sip:+1-212-555-1212:1234@gateway.com;user=phone | +1-212-555-1212@gateway.com |
| sips: 1212@gateway.com | 1212@gateway.com |
| sip:alice@192.0.2.4 | alice@192.0.2.4 |
| sip:atlanta.com;method=REGISTER?to=alice %40atlanta.com | atlanta.com |
| sip:alice; day=tuesday@atlanta.com | alice@atlanta.com |

### *Route Value*

The route value contains a list of peers and a peer selection mode attribute.

#### Peer Selection

- The peer selection mode attribute specifies how a peer in the peer list is selected. Available values are sequential and ratio.
- If the contained peers contain different transport types (ipproto), TCP, UDP, SCTP, only those peers that match the transport type of the originating connection is used for peer selection.
- In sequential mode, the peers are selected in the order listed. The first peer is used unless all of its members are down.
- In ratio mode, the ratio value in each peer shall be used to determine the distribution of message to each peer.
- Once a peer has been selected, a pool member from the peer's pool is selected based on the pool's lb-mode attribute.
- The peer's transport-config name (MR transport-config object refer to HighperformancemessageroutingframeworkforIMSprotocolsFS#Transport-config) is used to configure the type of connection (transport, security, protocol, rules, snat).

#### Host Selection

- If the selected peer does not contain a pool, the destination ip and port of the message's originating connection is used as the destination host.
- If the selected peer does not contain a transport-config name, the transport type and name of the message's originating connection is used as the destination host.

- If the selected peer contains a pool with no pool members, the message is returned to the originator marked as unroutable.
- If the selected peer contains a pool with pool members. one active pool member is selected as per the pool's specified load balancing mode.

### SIP Session Profile

This profile is attached to every virtual & associated with each peer of a routing instance. This profile has settings that can affect the SIP message processing. Multiple SIP session profiles can be in use in a single routing instance. The virtual/peer processes the ingress/egress messages per its sip-session profile settings.

```
ltm message-routing sip profile session sipsession {
    custom-via <string>
    defaults-from none
    do-not-connect-back <enabled|disabled>
    enable-sip-firewall <yes|no>
    generate-response-on-failure <enabled|disabled>
    honor-via <enabled|disabled>
    insert-record-route-header <enabled|disabled>
    insert-via-header <enabled|disabled>
    loop-detection <enabled|disabled>
    maintenance-mode <enabled|disabled>
    max-forwards-check <enabled|disabled>
    max-msg-header-count <integer>
    max-msg-header-size <integer>
    max-msg-size <integer>
    persistence {
        persist-key <Call-ID|Src-Addr|Custom>
        persist-timeout <integer>
        persist-type <session|none>
    }
}
```

The sip protocol profile has the following attributes.

**Table 8: SIP Session Profile Attributes**

| Attribute | Description | Type | Acceptable Values | Default |
|---|---|---|---|---|
| max-msg-size | Specifies the maximum acceptable SIP message size in bytes. The message that exceeds this size is silently discarded. | UINT32 | 1 to 4294967295 (32-bits) | 65535 |
| max-msg-header-count | Specifies the maximum count of expected header fields; The message that exceeds this limit is silently discarded. | UINT32 | 6 to 4096 | 64 |
| max-msg-header-size | Specifies the maximum message header size in bytes; The message that exceeds this limit is silently discarded. | UINT32 | 1 to 4294967295 (32-bits) | 16000 |
| generate-response-on-failure | Enables to send failure response messages such as 4xx, 5xx and 6xx, when a SIP request is being dropped; Note: Where it is specified "silently" discarded/dropped, no error response is generated. In any case, a dropped message (request/response) is tracked in appropriate statistics counter. | BOOL | Enabled/ Disabled | Disabled |

| Attribute | Description | Type | Acceptable Values | Default |
|---|---|---|---|---|
| Maintenance Mode | When selected (enabled), sends a SIP response of 503 Service Unavailable for an incoming SIP request. The SIP response to the SIP request is dropped. | BOOL | Enabled/ Disabled | Disabled |
| max-forwards-check | Enables check on max-forwards; If 0, the request is discarded. An error response is sent, if configured. | BOOL | Enabled/ Disabled | Enabled |
| loop-detection | Enables loop-detection check and in case loop detected, the request is discarded. An error response is sent, if configured.<br><br>***Note:*** *A request is detected as seen before (forwarded/spiraled/looped) only if self inserted Via is found in the message and the value of its branch param plays a key role in detecting loop versus spiral. Hence enabling via insertion becomes a requirement to do loop detection check.*<br><br>In ALG mode, Via header is not inserted by default and there is no loop detection in this mode. | BOOL | Enabled/ Disabled | Disabled |
| insert-via-header | Enables insertion of top Via; When enabled, custom params to help route the response back are inserted, along with sent-by field of Via. The source address:port of the flow forwarding the request is filled as value for sent-by field of Via unless user provides custom via value. The custom params inserted to help routing, helps improve performance as it facilitates routing without any lookup. The via is inserted at egress side of the flow, after SIP_REQUEST_SEND event. | BOOL | Enabled/ Disabled | LB MODE: Enabled<br><br>ALG MODE: disabled |
| custom-via | Specifies the custom value for the sent-by field of Via. Only the sent-by component value is mentioned here not the complete header. | STRING | <IP or FQDN name>[:<port>] | None |
| honor-via | Enables to honor via that is not inserted by a BIG-IP® system for routing the response. | BOOL | Enabled/ Disabled | LB MODE: Enabled<br><br>ALG MODE: disabled |
| insert-record-route-header | Enables insertion of record-route header in requests that establish dialog. When enabled, along with URI, the custom | BOOL | Enabled/ Disabled | Disabled |

| Attribute | Description | Type | Acceptable Values | Default |
|---|---|---|---|---|
| | params may be added to facilitate the routing of subsequent requests within this call to avoid route lookup. The record route URI is the local-IP & port of flows that are used for forwarding the message. | | | |
| sip-firewall | Enables application of firewall policy | BOOL | Enabled/ Disabled | Disabled |
| do-not-connect-back | Controls whether connection to a request originator is established (if it no longer exists) in order to deliver response. When disabled, responses that cannot be forward using an existing connection are dropped. | BOOL | Enabled/ Disabled | Disabled |
| persistence | | | | |
| persist-key | Specifies the method to extract the key value that is used to persist on.<br>• Call-ID - To persist based on the "Call-ID" header field value in the message.<br>• Src-Addr - To persist based on originating IP address in the message<br>• Custom - To persist based on the custom key specified using iRule. | ENUM | Call-ID/Src-Addr/Custom | Call-ID |
| persist-type | Specifies the type of the persistence to be used for the specified "persist-key" attribute value, the currently supported type is session.<br>• Session - Uses session DB for storage, no hash is applied. The key used for session DB is value specified in the "persist-key" attribute. Insert-via-header must be enabled when persist-type is set to "Session", if not a validation error is thrown.<br>• None - Persistence is disabled<br>• Persistence is not applicable for SIP ALG modes. | ENUM | Session/None | Session |
| persist-timeout | Indicates the timeout value of persistence entries in seconds.<br>It's recommended to have the persist-timeout to be greater than transaction timeout, specified in the SIP session configuration, as the lesser of the two is used when creating the persist record on receiving of the initial SIP request message. The initial SIP request can be | UINT32 | 1 to 4294967295 (32-bits) | 180 |

| Attribute | Description | Type | Acceptable Values | Default |
|---|---|---|---|---|
| | INVITE/SUBSCRIBE/MESSAGE. Upon receiving of the response for the initial SIP Request message the persistence record is updated with the persist-timeout value. (For any subsequent responses received the persist timeout is updated for the persist record.) | | | |

### Peer Object

A peer object is used to define a set of hosts and the the method to connect with them. Peers are used to create static routes. The peer structure is protocol independent while each protocol implementation of MRF will define its own static route structure.

```
ltm message-routing sip peer <named-object> {
  app-service <string>
  auto-initialization <enabled/disabled>
  auto-initialization-interval <integer>
  connection-mode <per-peer/per-tmm/per-blade/per-client>
  description <string>
  number-connections <integer>
  partition <string>
  pool <pool_name>
  ratio <integer>
  transport-config <tc_name>
}
```

*Peer Attributes*

**Table 9: Peer Attributes**

| Attribute | Description | Default |
|---|---|---|
| pool | Pool associated with the peer. If only one peer, then configure a single-member pool. If none, the message will be forwarded to the destination address and port of the originating connection. | none |
| transport-config | Specifies the transport-config that defines the parameters of the outgoing connection. If none, the parameters of the originating connection will be used to create the outgoing connection. | none |
| connection-mode | Specifies how the number of connections per peer are to be limited as follows: per-peer, per-blade, per-tmm, per-client.<br><br>If a transport config is not specified, the attributes of the originating connection of the message being routed will be used to create the outgoing connection. In this case, the connection-mode in the peer object will be ignored. | per-peer |
| number-connections | Specifies the number of connections between the BIG-IP® system and a peer.<br><br>If a transport config is not specified, the attributes of the originating connection of the message being routed will be used to create the outgoing connection. In this case, the number-connections in the peer object will be ignored. | 1 |
| ratio | Used to designate the ratio of this peer when used within a route with a peer-selection-mode of ratio. | 1 |

| Attribute | Description | Default |
|---|---|---|
| auto-initialization | If enabled, the BIG-IP® system will automatically create outbound connections to the active pool members in the specified pool using the configuration of the specified transport-config. For auto-initialization to attempt to create a connection, the peer must be included in a route that is attached to a router instance. For each router instance that the peer is contained in, a connection will be initiated. The auto-initialization logic will verify at a configurable interval if the a connection exists between the BIG-IP system and the pool members of the pool. If a connection does not exist, it will attempt to reestablish one. | disabled |
| auto-initialization-interval | Specifies the interval (in milliseconds) that attempts to initiate a connection occur. Valid ranges are from 500ms to 65535ms | 5000ms |

### *Connection Modes*

#### Per Peer

A BIG-IP® system will make just one connection to a peer. This means that only one TMM is connected to each Peer. While this connection mode uses fewer connections it will introduce latency. This will happen when messages are disaggregated to the wrong TMM and must be forwarder. The following diagram provides additional detail.

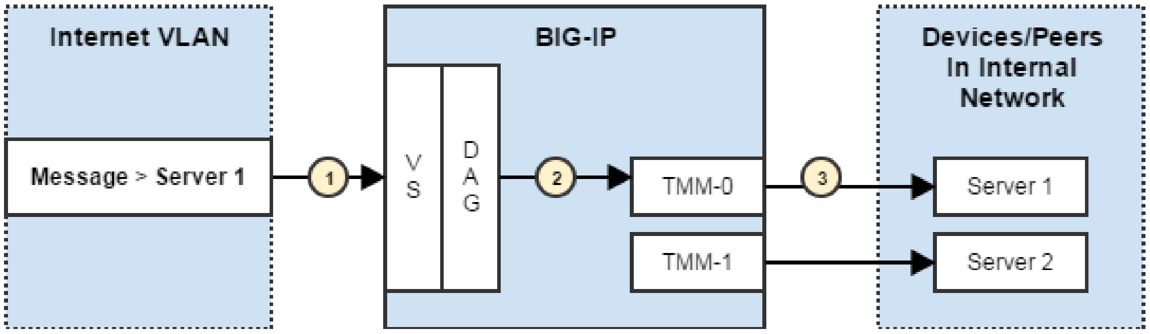


**Figure 2: Optimum scenario**

1. Message arrives on a Virtual Server (VS)
2. Message is disaggregated to TMM-0
3. TMM-0 is connected to the correct server so the message is sent

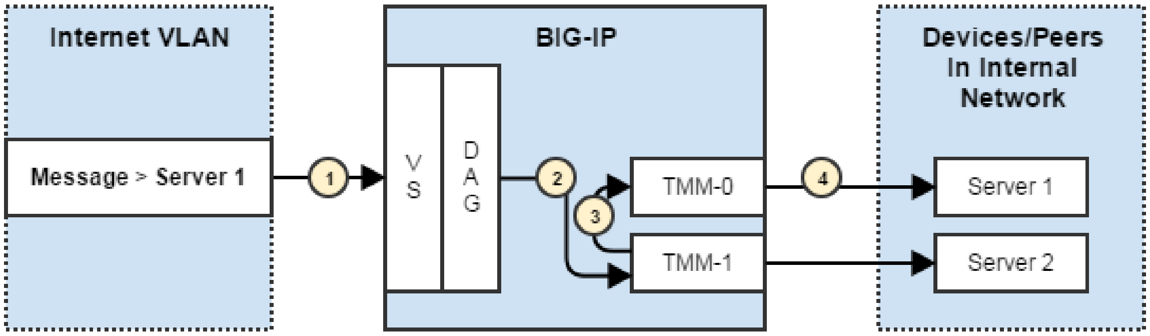


**Figure 3: Performance impacted scenario**

1. Message arrives on a Virtual Server (VS)
2. Message is disaggregated to TMM-1

3. TMM-1 is not connected to Server 1 so message must be forwarded to the correct TMM. This will introduce latency.
4. TMM-0 is send message to Server 1

### Per TMM

A BIG-IP® system will make a connection from every TMM to the same peer. This means a machine with 8 cores will have 8 connections per peer. While this increases the number of active connections, it also improves performance because there is no need to forward messages between TMMs.

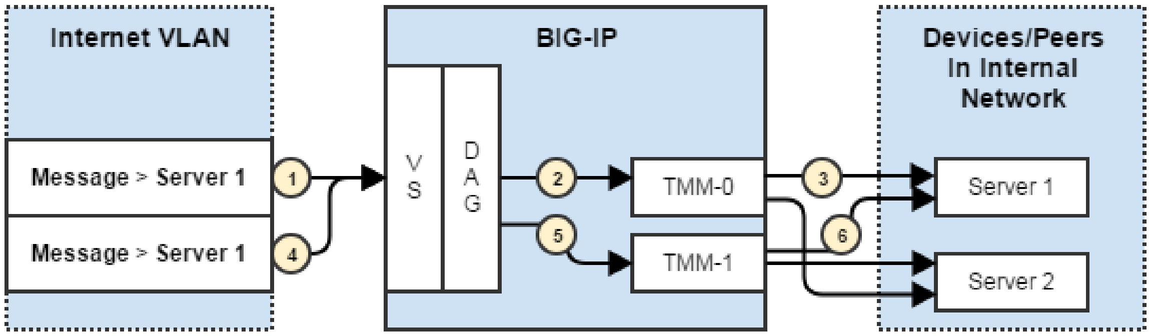


**Figure 4: Every TMM is connected to every peer which decreases latency but increases the number of connections**

1. Message arrives on a Virtual Server (VS)
2. Message is disaggregated to TMM-0
3. TMM-0 is connected to the correct server so the message is sent
4. Second message arrives
5. Message is disaggregated to TMM-1
6. TMM-1 is connected to the requested server so the message can be sent directly

### Per Blade

A BIG-IP® system creates one connection per blade to each peer. This provides a balanced performance approach between the per peer connection mode (only one connection) and a per tmm connection mode (a connection from each TMM). This mode only makes sense for a hardware chassis with multiple blades.

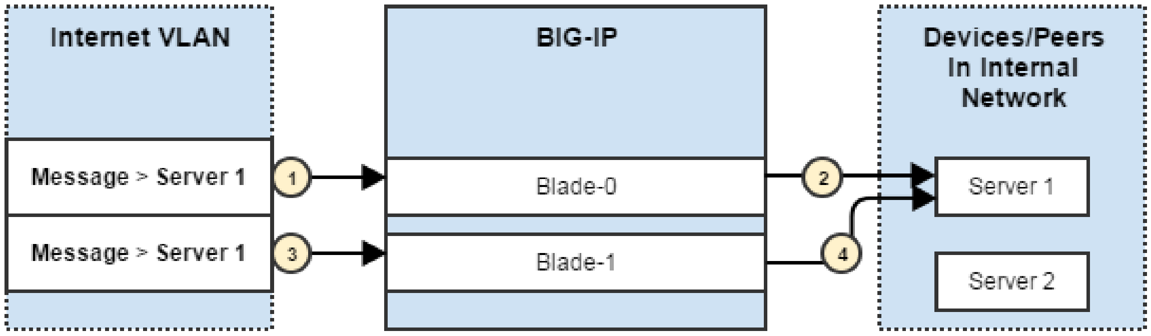


**Figure 5: Each blade will make a single connection to each peer.**

1. Message arrives on Blade 0
2. Blade 0 opens a connection to Server 1 and forwards the message
3. Second message arrives
4. Blade 1 opens a connection to server 1 and forwards the message

***Note:*** *A connection will not be opened to Server 2 until a message targeted at that server arrives.*

### Transport Config

A transport config defines the parameters of a new outgoing connection.

The behavior of a transport-config is determined by the set of profiles attached to it. To configure a MRF SIP transport-config a transport profile (TCP, UDP or SCTP), and a sipsession profile must be attached. The siprouter profile will be inherited by the router instance that creates the outgoing connection.

The behavior of a transport-config can further be extended by assigning iRules to it.

```
ltm message-routing sip transport-config <transport-1> {
   ip-protocol <tcp/udp/sctp/...>
   profiles {
      tcp {}
      diameter_protocol_test {}
   }
   source-address-translation {
      type automap
   }
   rules {
      some_irule
   }
   source-port <integer>
}
```

*Transport Config Attributes*

**Table 10: Transport Config Attributes**

| Attribute | Description | Default |
|---|---|---|
| ip-protocol | Specifies the ip protocol. This will be automatically set based on the transport profile added. This value is read-only. | none |
| source-port | Specifies the source port to be used for the connection being created. If the source-port is zero, an empirical port will be used. | 0 |
| profiles | The transport protocol and the protocol-specific profile associated with this outgoing connection. | |
| source-address-translation | Specifies the source-address-translation type and the pool. | |
| rules | List of iRules associated with this outgoing connection. | none |

#### *Source Address Translation*

**Table 11: Source Address Translation**

| Sub-Attribute | Description | Default |
|---|---|---|
| type | Specifies the type of source address translation to perform | automap |
| pool | Specifies the name of the snap pool | none |

#### *Source Address Translation Types*

**Table 12: Source Address Translation Types**

| Type | Description |
|---|---|
| automap | The self-ip of the outgoing vlan will be used as the source address of the outgoing connection. |

| Type | Description |
|---|---|
| snat | A source address will be selected from the named snat pool |
| none | No source address translation will be performed. |

## ALG without SNAT (No Address Translation)

- Snoop control messages flowing through to manage media flows.
- iRule can be used to rewrite headers.
- Create media records in session db.
- Create deny listeners to drop media packets received before the callee responds with its media details.
- Create media flows to forward packets between caller and callee.
- High Availability (HA)
  - Call table replication (supports failback)
  - Control connection mirroring (can be recreated on failback by endpoint)
  - Media flow mirroring

### Configuration Objects

#### Virtual Server

A virtual server is a traffic-management object on the BIG-IP® system that is represented by an IP address and a service. Clients on an external network can send application traffic to a virtual server, which then directs the traffic according to your configuration instructions.

The virtual server configuration contains a destination address and mask which specifies what IP addresses and port the virtual server will listen for incoming packets. The virtual server object also contains a source address allowing it to limit packets to those packets that originate from a range of devices.

The behavior of a virtual server is determined by the set of profiles attached to the virtual server. To configure a MRF SIP virtual server, transport profile (TCP, UDP or SCTP), a sipsession-alg profile and a siprouter-alg profile must be attached to the virtual server.

The behavior of a virtual can further be extended by assigning iRules to the virtual.

All virtual servers that share the same siprouter profile will share the same router instance. The routing instance owns the route table, the persistence table and flowmap table (a table of all open connection usable for message forwarding).

- A virtual server of type "message-routing" assigned with "session" & "router" profile is part of a SIP router-instance.
- MRF SIP ALG does not require persistence or message routing.
- Persistence is configured through the SIP session profile, hence a persist profile must not be attached to the virtual server.
- One or more iRules can be attached. The rules are validated against the configured transport and the events/commands as supported by SIP and MRF.
- MRF SIP ALG without source-address-translation does not support source-address-translation. The virtual's source-address-translation type must be set to none

Please note that profile call statistics in this mode will double-count hairpinned calls.

```
ltm virtual <virtual name> {
    cmp-enabled yes
    destination <ip address>:<port>
    enabled | disabled
    ip-protocol <tcp|udp|sctp>
    mask <netmask>
    profiles {
        siprouter-alg
        sipsession-alg
        <tcp|udp|sctp>
        <other profiles>
    }
    rules <list of iRules|none>
    source 0.0.0.0/0
    source-address-translation {
        type none
    }
    vlans {
        <list of vlans>
    }
    vlans-enabled | vlans-disabled
}
```

### SIP Router Profile

```
ltm message-routing sip profile router siprouter-alg {
    inherited-traffic-group <true|false>
    max-pending-bytes <integer>
    max-pending-messages <integer>
    media-proxy {
        max-media-sessions <integer>
        media-inactivity-timeout <integer>
    }
    mirror <enabled|disabled>
    mirrored-message-sweeper-interval 1000
    operation-mode application-level-gateway
    routes none
    session {
        max-session-timeout <integer>
        transaction-timeout <integer>
    }
    traffic-group <traffic group name>
    use-local-connection enabled
}
```

**Table 13: SIP Router Profile Attributes**

| Attribute | Description | Type | Acceptable Values | Default |
|---|---|---|---|---|
| operation-mode | Sets the operation mode of the SIP routing instance.<br>Must be set to application-level-gateway. | ENUM | load-balancing<br>application-level-gateway | load-balancing |
| routes | N/A in ALG Mode | sip-route | A list of sip-route objects | |
| max-pending-messages | The maximum number of pending messages that are held while waiting for a connection to a peer to be created. Once reached any additional messages to the peer is flagged as undeliverable and returned to the originator. | UINT32 | 1 to 4294967295 (32-bits) | 23768 |
| max-pending-bytes | The maximum number of bytes contained within pending messages that is held while waiting for a connection to a peer | UINT32 | 1 to 4294967295 (32-bits) | 64 |

| Attribute | Description | Type | Acceptable Values | Default |
|---|---|---|---|---|
| | to be created. Once reached any additional messages to the peer is flagged as undeliverable and returned to the originator. | | | |
| use-local-connection | N/A in ALG Mode | BOOL | Enabled/ Disabled | Enabled |
| traffic-group | The traffic group for the router instance. All virtual servers using this router profile will have the their traffic group replaced by the traffic group of the router profile. | traffic-group | | default (inherited from partition) |
| mirror | Enables mirroring of all incoming connections for all virtual servers using this router instance, and all outgoing connection created by this router instance. | BOOL | enabled/ disabled | disabled |
| mirrored-message-sweeper-interval | This attribute sets the frequency of the mirrored message sweeper. For virtual servers where mirroring is enabled, the received messages will be processed on both the active device and the standby device. On the standby device, the messages are not routed, instead they are stored in a message store until the active device sends a notification that the message has been routed to the standby device so that the standby device can deliver the message to the equivalent connection for egress processing. A sweeper has been implemented to drop messages from the message store if they remain in the store longer than the time specified in this attribute. The time shall be in milliseconds. | UINT32 | | 1000(ms) |
| media-proxy | | | | |
| max-media-sessions | This attribute is valid when the operation-mode is application-level-gateway. Specifies the maximum number of media sessions that are allowed per call. | UINT32 | 1 to 10 (32-bits) | 6 |
| media-inactivity-timeout | This attribute is valid when the operation-mode is application-level-gateway. Specifies the maximum duration (in seconds) that a media flow is active with no | UINT32 | 1 to 120 (32-bits) | 120 |

| Attribute | Description | Type | Acceptable Values | Default |
|---|---|---|---|---|
| | RTP packets. After this period the RTP flow is removed. This timeout is applicable only to RTP packet where as the RTCP packet will have the timeout set to the max-session-timeout. | | | |
| session | | | | |
| dialog-establishment-timeout | This attribute is valid when the operation-mode is application-level-gateway. Specifies the timeout (in seconds) that represents the Timer B as per RFC 3261, the INVITE transaction timeout. The dialog-establishment-timeout is used by the Call Table. The default value is 32 seconds. | UINT32 | 1 to 4294967295 (32-bits) | 32 |
| max-session-timeout | This attribute is valid when the operation-mode is application-level-gateway. Specifies the maximum duration (in seconds) that a call and its media remains active. After this period the call and its media is terminated. | UINT32 | 1 to 4294967295 (32-bits) | 7200 |
| transaction-timeout | Specifies the maximum time in seconds between request and its response. A provisional response restarts the timer. This may not affect all transactions. The scenarios where BIG-IP waits for response (like final response for REGISTER request), is impacted, by dropping any persistent data maintained for this request. | UINT32 | 1 to 4294967295 (32-bits) | 180 |

### *Operation Mode*

**Table 14: SIP operation mode**

| operation-mode | Description |
|---|---|
| load-balancing | Configures the SIP routing instance to operate in load-balancing mode. See *How to configure DAG Modes* for details. |
| application-level-gateway | Configures the SIP routing instance to operate in application level gateway mode (ALG). See *Default DAG* for details |

### *SIP Session Profile*

This profile is attached to every virtual & associated with each peer of a routing instance. This profile has settings that can affect the SIP message processing. Multiple SIP session profiles can be in use in a single routing instance. The virtual/peer processes the ingress/egress messages per its sip-session profile settings.

```
ltm message-routing sip profile session sipsession {
    custom-via <string>
    defaults-from none
    do-not-connect-back <enabled|disabled>
    enable-sip-firewall <yes|no>
    generate-response-on-failure <enabled|disabled>
    honor-via <enabled|disabled>
    insert-record-route-header <enabled|disabled>
    insert-via-header <enabled|disabled>
    loop-detection <enabled|disabled>
    maintenance-mode <enabled|disabled>
    max-forwards-check <enabled|disabled>
    max-msg-header-count <integer>
    max-msg-header-size <integer>
    max-msg-size <integer>
    persistence {
        persist-key <Call-ID|Src-Addr|Custom>
        persist-timeout <integer>
        persist-type <session|none>
    }
}
```

The sip protocol profile has the following attributes.

**Table 15: SIP Session Profile Attributes**

| Attribute | Description | Type | Acceptable Values | Default |
|---|---|---|---|---|
| max-msg-size | Specifies the maximum acceptable SIP message size in bytes. The message that exceeds this size is silently discarded. | UINT32 | 1 to 4294967295 (32-bits) | 65535 |
| max-msg-header-count | Specifies the maximum count of expected header fields; The message that exceeds this limit is silently discarded. | UINT32 | 6 to 4096 | 64 |
| max-msg-header-size | Specifies the maximum message header size in bytes; The message that exceeds this limit is silently discarded. | UINT32 | 1 to 4294967295 (32-bits) | 16000 |
| generate-response-on-failure | Enables to send failure response messages such as 4xx, 5xx and 6xx, when a SIP request is being dropped; Note: Where it is specified "silently" discarded/dropped, no error response is generated. In any case, a dropped message (request/response) is tracked in appropriate statistics counter. | BOOL | Enabled/ Disabled | Disabled |
| Maintenance Mode | When selected (enabled), sends a SIP response of 503 Service Unavailable for an incoming SIP request. The SIP response to the SIP request is dropped. | BOOL | Enabled/ Disabled | Disabled |
| max-forwards-check | Enables check on max-forwards; If 0, the request is discarded. An error response is sent, if configured. | BOOL | Enabled/ Disabled | Enabled |

| Attribute | Description | Type | Acceptable Values | Default |
|---|---|---|---|---|
| loop-detection | Enables loop-detection check and in case loop detected, the request is discarded. An error response is sent, if configured.<br><br>**Note:** *A request is detected as seen before (forwarded/spiraled/looped) only if self inserted Via is found in the message and the value of its branch param plays a key role in detecting loop versus spiral. Hence enabling via insertion becomes a requirement to do loop detection check.*<br><br>In ALG mode, Via header is not inserted by default and there is no loop detection in this mode. | BOOL | Enabled/ Disabled | Disabled |
| insert-via-header | Enables insertion of top Via; When enabled, custom params to help route the response back are inserted, along with sent-by field of Via. The source address:port of the flow forwarding the request is filled as value for sent-by field of Via unless user provides custom via value. The custom params inserted to help routing, helps improve performance as it facilitates routing without any lookup. The via is inserted at egress side of the flow, after SIP_REQUEST_SEND event. | BOOL | Enabled/ Disabled | LB MODE: Enabled<br><br>ALG MODE: disabled |
| custom-via | Specifies the custom value for the sent-by field of Via. Only the sent-by component value is mentioned here not the complete header. | STRING | <IP or FQDN name>[:<port>] | None |
| honor-via | Enables to honor via that is not inserted by a BIG-IP® system for routing the response. | BOOL | Enabled/ Disabled | LB MODE: Enabled<br><br>ALG MODE: disabled |
| insert-record-route-header | Enables insertion of record-route header in requests that establish dialog. When enabled, along with URI, the custom params may be added to facilitate the routing of subsequent requests within this call to avoid route lookup. The record route URI is the local-IP & port of flows that are used for forwarding the message. | BOOL | Enabled/ Disabled | Disabled |
| sip-firewall | Enables application of firewall policy | BOOL | Enabled/ Disabled | Disabled |

| Attribute | Description | Type | Acceptable Values | Default |
|---|---|---|---|---|
| do-not-connect-back | Controls whether connection to a request originator is established (if it no longer exists) in order to deliver response. When disabled, responses that cannot be forward using an existing connection are dropped. | BOOL | Enabled/ Disabled | Disabled |
| persistence | | | | |
| persist-key | Specifies the method to extract the key value that is used to persist on.<br>• Call-ID - To persist based on the "Call-ID" header field value in the message.<br>• Src-Addr - To persist based on originating IP address in the message<br>• Custom - To persist based on the custom key specified using iRule. | ENUM | Call-ID/Src-Addr/Custom | Call-ID |
| persist-type | Specifies the type of the persistence to be used for the specified "persist-key" attribute value, the currently supported type is session.<br>• Session - Uses session DB for storage, no hash is applied. The key used for session DB is value specified in the "persist-key" attribute. Insert-via-header must be enabled when persist-type is set to "Session", if not a validation error is thrown.<br>• None - Persistence is disabled<br>• Persistence is not applicable for SIP ALG modes. | ENUM | Session/None | Session |
| persist-timeout | Indicates the timeout value of persistence entries in seconds.<br>It's recommended to have the persist-timeout to be greater than transaction timeout, specified in the SIP session configuration, as the lesser of the two is used when creating the persist record on receiving of the initial SIP request message. The initial SIP request can be INVITE/SUBSCRIBE/MESSAGE. Upon receiving of the response for the initial SIP Request message the persistence record is updated with the persist-timeout value. (For any subsequent responses received the persist timeout is updated for the persist record.) | UINT32 | 1 to 4294967295 (32-bits) | 180 |

# Disaggregation (DAG) Modes

## Disaggregation (DAG) Modes

### How to configure DAG Modes

**Table 16: How to configure DAG Modes**

| DAG Mode | Configuration Object | TMSH Commands |
|---|---|---|
| Default-DAG | VLAN | `$ modify net vlan <vlan_name> cmp-hash default` |
| SP-DAG | VLAN | `$ modify net vlan <src_vlan_name> cmp-hash src-ip`<br>`$ modify net vlan <dst_vlan_name> cmp-hash dst-ip` |
| RR-DAG | VLAN | `$ modify net vlan <vlan_name> dag-round-robin enabled`<br>`$ modify sys db dag.roundrobin.udp.portlist value "5060"`<br>`$ modify ltm profile udp <udp_profile_name> idle-timeout 0` |

### Default DAG

The Default DAG uses a hash of source and destination port. It is useful when ephemeral ports are used in client side and server side connections. When source and destination ports are the same TMM-0 will be used. This is an issue in that the traffic will not be load balanced and TMM-0 will quickly be overloaded. This DAG requires randomness in the source or destination port. If a client doesn't specify a source port then an ephemeral port will be used and Default DAG will work properly. Note, the ephemeral port must increment randomly or by single digits. If it's incremented by an even number, such as two, or by the number of TMMs then it's possible that it will hash to the same TMM or a small set of TMMs, which will negatively impact BIG-IP® system performance.

#### Key Points

- Port Based.
- Works best when clients use ephemeral ports.
- Can work with 1 to n clients.

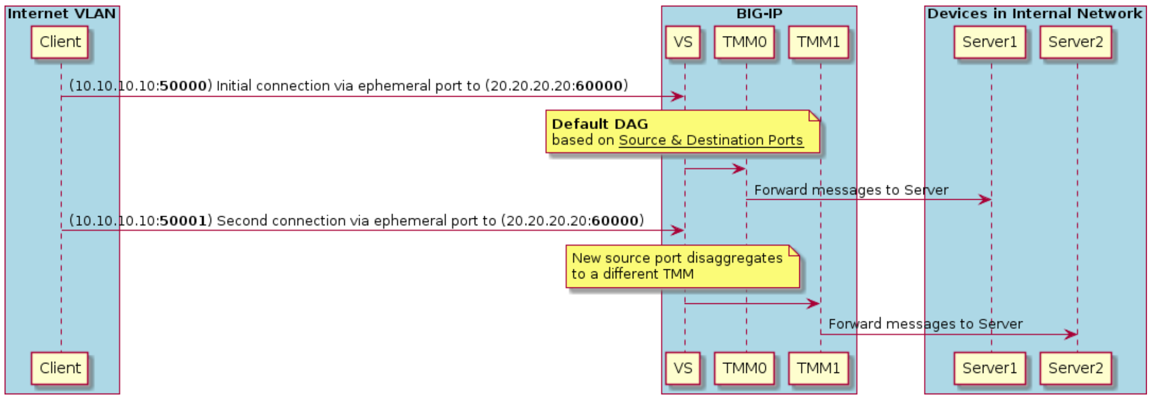

Internet VLAN
Client
BIG-IP
VS
TMM0
TMM1
Devices in Internal Network
Server1
Server2
(10.10.10.10:50000) Initial connection via ephemeral port to (20.20.20.20:60000)
Default DAG based on Source & Destination Ports
Forward messages to Server
(10.10.10.10:50001) Second connection via ephemeral port to (20.20.20.20:60000)
New source port disaggregates to a different TMM
Forward messages to Server

**Figure 6: Default DAG example**

## Source/Destination DAG (SP-DAG)

The SP-DAG uses a hash of source IP (from client) and destination IP address (server). This mode should be used when source and destination ports are hardcoded (for example 5060). In that case, a BIG-IP® system requires multiple client IP address or multiple server IP addresses. Keep in mind, most connections are initiated by the client and that's the "Source DAG" option. In this case, the "Destination DAG" could be a single IP, but the source client IP should have more that a single IP address.

### Key Points

- IP Address Based
- Works best when number of clients is equal to or more than the number of TMMs in BIG-IP® system.
- Performance will be impacted if clients consist of only a few SIP Proxy connections. In this case the IP Address entropy will be too low to load balance the incoming packets across available TMMs.

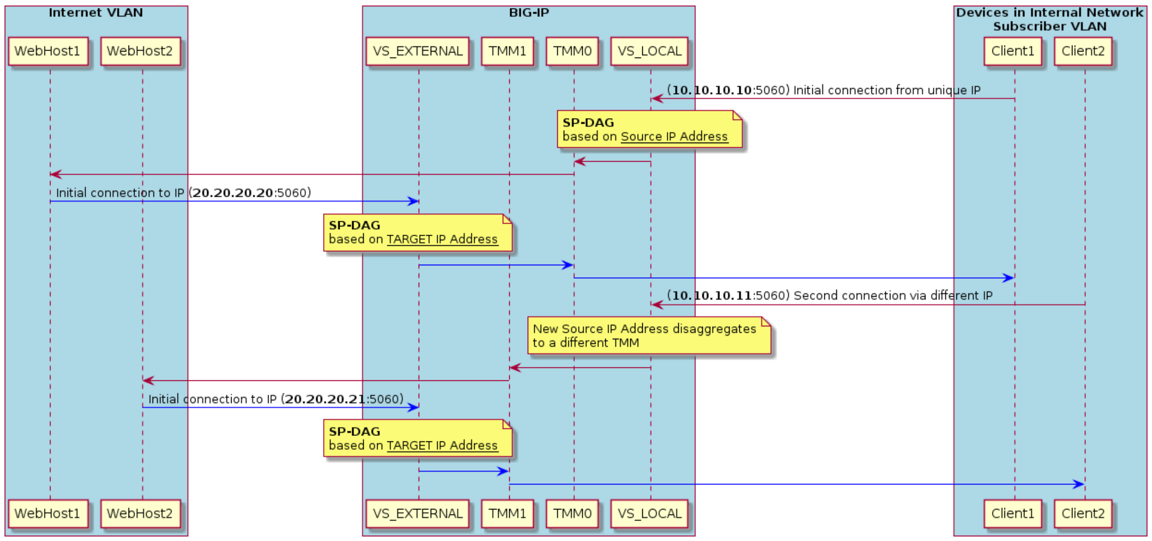

Internet VLAN
WebHost1
WebHost2
BIG-IP
VS_EXTERNAL
TMM1
TMM0
VS_LOCAL
Devices in Internal Network Subscriber VLAN
Client1
Client2
(10.10.10.10:5060) Initial connection from unique IP
SP-DAG based on Source IP Address
Initial connection to IP (20.20.20.20:5060)
SP-DAG based on TARGET IP Address
(10.10.10.11:5060) Second connection via different IP
New Source IP Address disaggregates to a different TMM
Initial connection to IP (20.20.20.21:5060)
SP-DAG based on TARGET IP Address

**Figure 7: Source/Destination DAG (SP-DAG) example**

## Round Robin DAG (RR-DAG)

RR-DAG was designed to overcome the low entropy limitations of Default DAG and SP-Dag; although for UDP only. Furthermore, RR-DAG is hardware only and can't be used in a VE. Round Robin DAG distributes traffic by sending each consecutive packet to a different TMM. It does not rely on the IP address, or source port, of the client. The Round Robin DAG is configured on a per-VLAN basis.

### Key Points

- UDP Only
- Requires hardware (not an option in VE)
- Sends each consecutive packet to a different TMM.

```
$ modify net vlan <vlan_name> dag-round-robin enabled
```

```
$ modify sys db dag.roundrobin.udp.portlist value "5060"
```

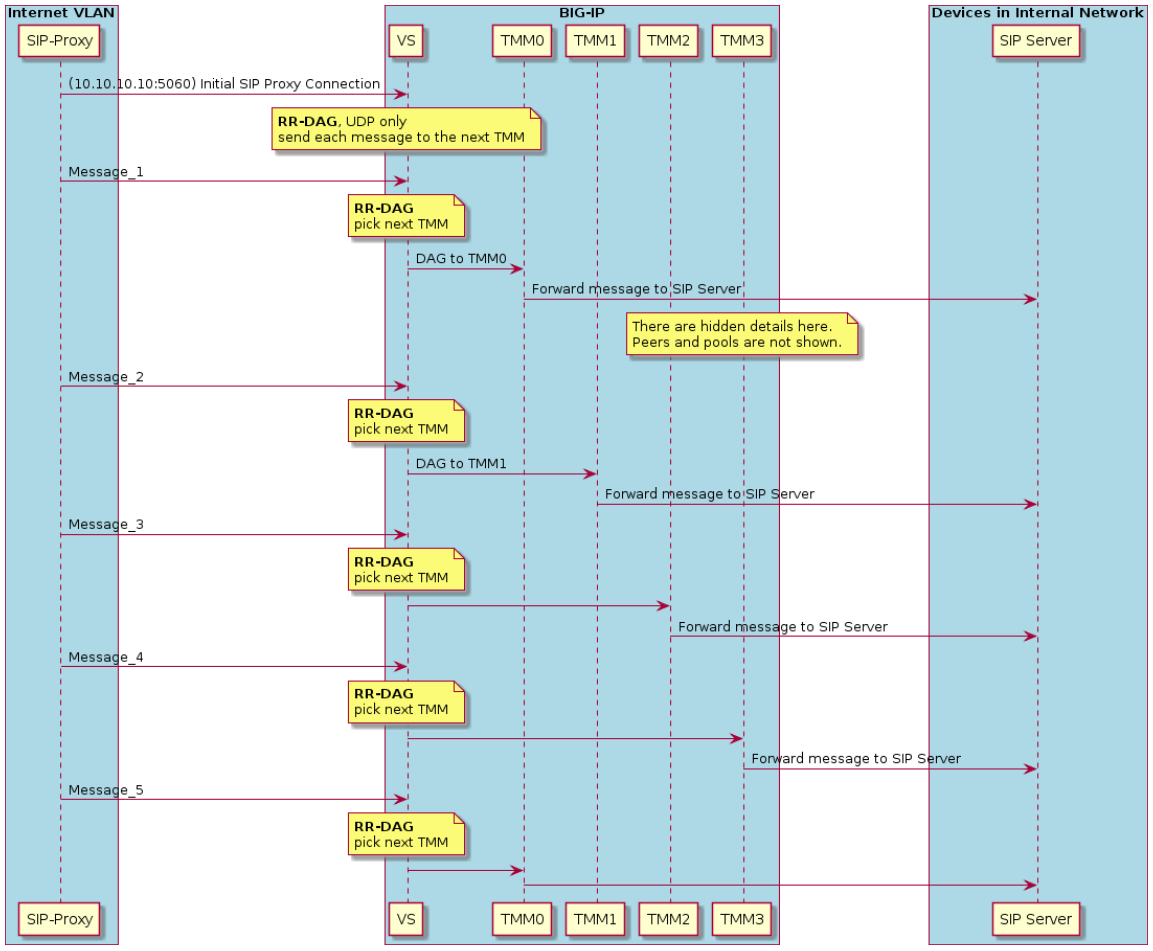

Internet VLAN
BIG-IP
Devices in Internal Network
SIP-Proxy
VS
TMM0
TMM1
TMM2
TMM3
SIP Server
(10.10.10.10:5060) Initial SIP Proxy Connection
RR-DAG, UDP only
send each message to the next TMM
Message_1
RR-DAG
pick next TMM
DAG to TMM0
Forward message to SIP Server
There are hidden details here.
Peers and pools are not shown.
Message_2
RR-DAG
pick next TMM
DAG to TMM1
Forward message to SIP Server
Message_3
RR-DAG
pick next TMM
Forward message to SIP Server
Message_4
RR-DAG
pick next TMM
Forward message to SIP Server
Message_5
RR-DAG
pick next TMM

**Figure 8: Round Robin DAG (RR-DAG) example**

# Deployment Use Cases

## Deployment Use Cases

### Basic Load Balancing (LB)

The "load-balancing" mode is used for scaling of capacity and/or providing high availability for SIP signaling servers/proxies/gateways. It allows steering of SIP signaling traffic to a pool of servers based on static SIP routes to spread the load over members of the pool. It provides Call-ID based load balancing persistence. It is the default mode of operation and does not automatically handle media flows. BIG-IP Via header is inserted by default for request messages and removed from response messages. Lasthop information added to the Via header information is used for response routing and route lookup is skipped for response messages.

In the load balancing operation mode, related media flows are not handled. The media flows associated with the SIP signaling message are assumed to be routed via other devices or virtual servers.

The configurations in sip session profile: Insert-via, Custom-via, Honor-via and Do-not-connect-back are inter-related.

#### Basic LB Example

In this example, you can see that a BIG-IP® system is adding and removing the top most Via such that the message will return to the BIG-IP before being forwarded out to the caller.

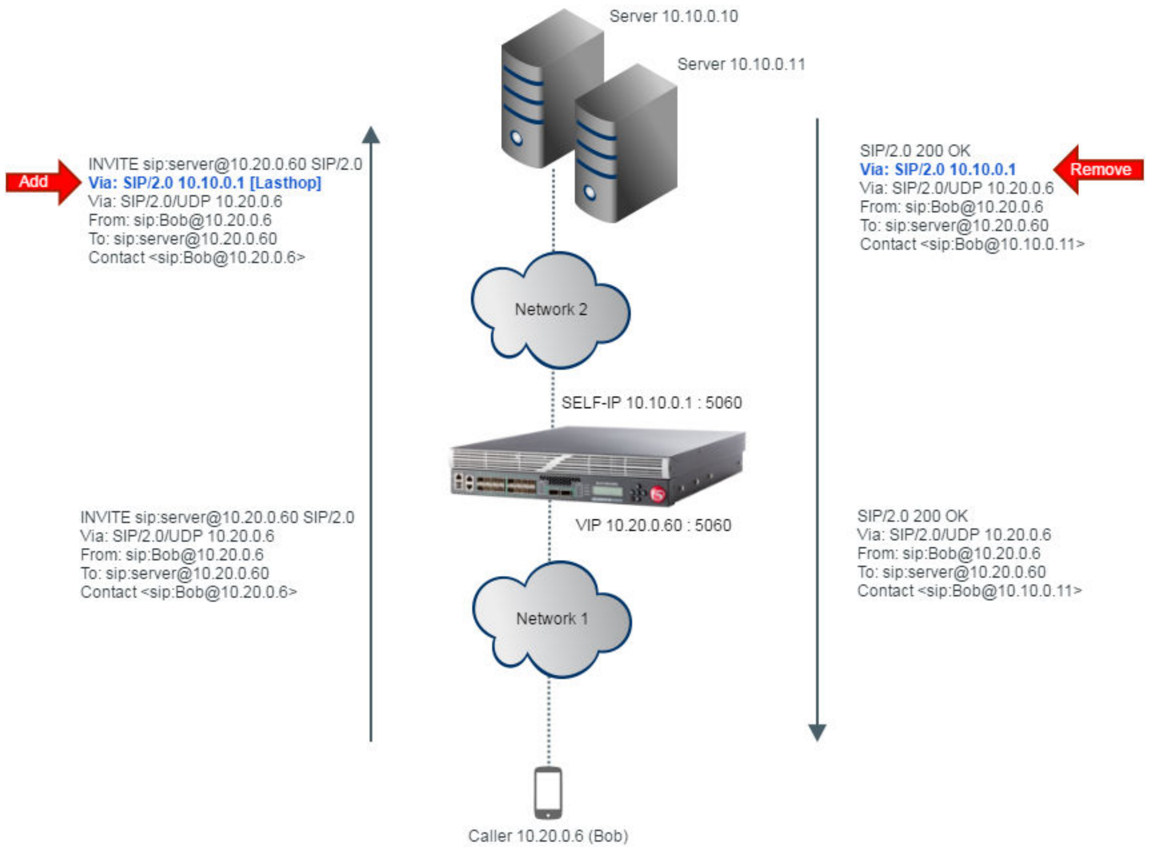

Server 10.10.0.10
Server 10.10.0.11
Add
INVITE sip:server@10.20.0.60 SIP/2.0
Via: SIP/2.0 10.10.0.1 [Lasthop]
Via: SIP/2.0/UDP 10.20.0.6
From: sip:Bob@10.20.0.6
To: sip:server@10.20.0.60
Contact <sip:Bob@10.20.0.6>
SIP/2.0 200 OK
Via: SIP/2.0 10.10.0.1
Remove
Via: SIP/2.0/UDP 10.20.0.6
From: sip:Bob@10.20.0.6
To: sip:server@10.20.0.60
Contact <sip:Bob@10.10.0.11>
Network 2
SELF-IP 10.10.0.1 : 5060
INVITE sip:server@10.20.0.60 SIP/2.0
Via: SIP/2.0/UDP 10.20.0.6
From: sip:Bob@10.20.0.6
To: sip:server@10.20.0.60
Contact <sip:Bob@10.20.0.6>
VIP 10.20.0.60 : 5060
SIP/2.0 200 OK
Via: SIP/2.0/UDP 10.20.0.6
From: sip:Bob@10.20.0.6
To: sip:server@10.20.0.60
Contact <sip:Bob@10.10.0.11>
Network 1
Caller 10.20.0.6 (Bob)

**Figure 9: Basic load balancing example**

### Configuration

In a route, at least wildcard entries for UDP and TCP default routes must be specified.

```
Route entry - ["", "", "", peer-udp] (Wildcard entry, default route for UDP)
Route entry - ["", "", "", peer-tcp] (Wildcard entry, default route for TCP)
```

### Load Balancing Configuration

```
ltm pool udp-default-pool {
     members {
         10.10.0.10:5060 {
            address 10.10.0.10
         }
         10.10.0.11:5060
            address 10.10.0.11
         }
     }
}

ltm message-routing sip peer peer-udp {
    pool udp-default-pool
}

ltm message-routing sip route default-route-udp {
    peers {  peer-udp  }
}

ltm message-routing sip profile router siprouter-lb {
    routes {
       default-route-udp
    }
}
 ltm virtual sip-lb-udp {
     destination 10.20.0.60:5060
     mask 255.255.255.255
     ip-protocol udp
     profiles {
         udp
         sipsession
         siprouter-lb
     }
}
```

## Load Balancing with Persistence

Persistence is configured through attributes in the "session" profile. Attaching a persistence profile to Virtual server is an invalid configuration and results in a configuration error.

There are two persistence types (persist-types) available; session (default) and none. If persist-type is set to "none", persistence is disabled. When persist-type is set to session, the persist-key specifies if BIG-IP persists on Call-ID (the default value), Src-Addr (source address), or custom. Persistence records are kept in SessionDB and therefore synchronized between TMMs and blades. Custom persist-key is specifically for iRules to create customer specific persistence keys. An iRule script may modify the message's persist-key during the SIP_REQUEST, SIP_RESPONSE or MR_INGRESS events. The value of the message's persist key after MR_INGRESS event is used for persistence lookup, if the "persist-key" is set to "Custom".

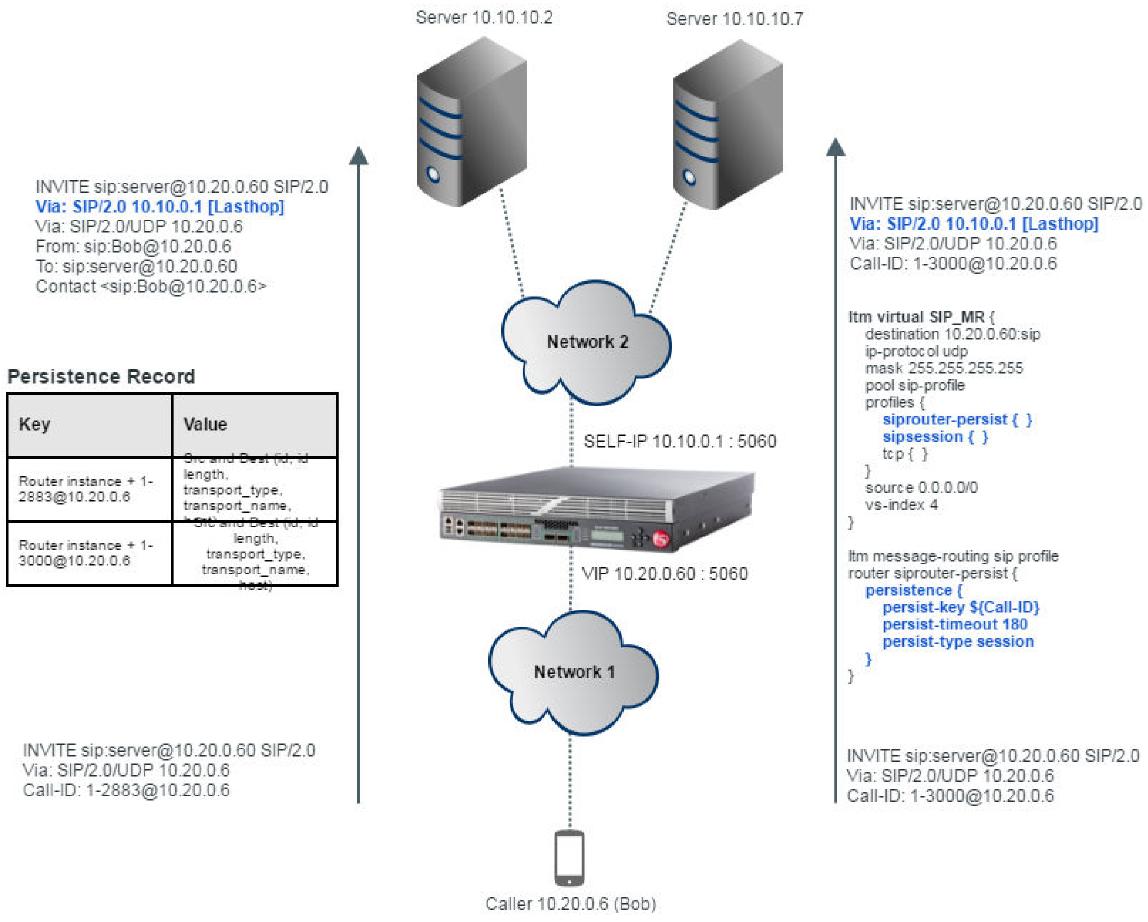
### Session Persistence

- Session persistence avoids a route lookup based on state recorded on the BIG-IP.
- It guarantees those messages carrying the same persistence key are going to be delivered to the same (L4) peer.

- Persistence entries are keyed by a value extracted from a message initiating a new session.
- The value used for the persist entry key, depends on the “persist-key” configuration attribute.
- When an existing persistence record is matched, the current message is delivered to the same (L4) destination avoiding a route lookup and LB pick.
- It is recommended to have the persist-timeout set to be greater than the transaction timeout, specified in the SIP session configuration, as the lesser of the two is used when creating the persist record on receiving of the initial SIP request message. The initial SIP request can be INVITE/SUBSCRIBE/ MESSAGE. Upon receiving of the response for the initial SIP Request message the persistence record is updated with the persist-timeout value. (For any subsequent responses received the persist timeout is updated for the persist record.)

### Basic LB with Session Persistence Example

This diagram shows a call from Call-ID 1-2883 @10.20.0.2 being load balanced to Server 10.10.10.2 and a call from 1-3000@10.20.0.6 being load balanced to 10.10.10.7 and the persist records created from these calls when persistence is enabled with a key of Call-ID.



```
INVITE sip:server@10.20.0.60 SIP/2.0
Via: SIP/2.0 10.10.0.1 [Lasthop]
Via: SIP/2.0/UDP 10.20.0.6
From: sip:Bob@10.20.0.6
To: sip:server@10.20.0.60
Contact <sip:Bob@10.20.0.6>
```

Persistence Record

| Key | Value |
|---|---|
| Router instance + 1-2883@10.20.0.6 | Src and Dest (id, id length, transport_type, transport_name, host) |
| Router instance + 1-3000@10.20.0.6 | Src and Dest (id, id length, transport_type, transport_name, host) |

```
INVITE sip:server@10.20.0.60 SIP/2.0
Via: SIP/2.0/UDP 10.20.0.6
Call-ID: 1-2883@10.20.0.6
```

```
INVITE sip:server@10.20.0.60 SIP/2.0
Via: SIP/2.0 10.10.0.1 [Lasthop]
Via: SIP/2.0/UDP 10.20.0.6
Call-ID: 1-3000@10.20.0.6
```

```
ltm virtual SIP_MR {
   destination 10.20.0.60:sip
   ip-protocol udp
   mask 255.255.255.255
   pool sip-profile
   profiles {
      siprouter-persist { }
      sipsession { }
      tcp { }
   }
   source 0.0.0.0/0
   vs-index 4
}

ltm message-routing sip profile
router siprouter-persist {
   persistence {
      persist-key ${Call-ID}
      persist-timeout 180
      persist-type session
   }
}
```

```
INVITE sip:server@10.20.0.60 SIP/2.0
Via: SIP/2.0/UDP 10.20.0.6
Call-ID: 1-3000@10.20.0.6
```

**Figure 10: Basic load balancing with session persistence example**

### Configuration

### Load Balancing Configuration

```
ltm pool session-pool {
     members {
          10.10.10.2:5060 {
              address 10.10.10.2
          }
          10.10.10.7:5060 {
              address 10.10.10.7
          }
     }
}

ltm message-routing sip peer sip-session-peer {
    pool session-pool
}

ltm message-routing sip route sip-session-route {
    peers {  sip-session-peer  }
}

ltm message-routing sip profile session sip-profile-1 {
   defaults-from-sipsession
   persistence {
       persist-type session
       persist-key Call-ID
       persist-timeout 30
   }
}

ltm message-routing sip profile router siprouter-persist {
    operation-mode load-balancer
    routes {
       sip-session-route
    }
    session {
       transaction-timeout 10
    }
}

ltm virtual sipmr-persist-session {
     destination 10.20.0.60:5060
     message-routing
     profiles {
         udp
         sip-profile-1
         sip-router-persist
     }
}
```

## SIP ALG without Address Translation

In Application Level Gateway (ALG) operation mode, the system will create media flows based on SDP offer/answer SIP message. The callee may begin sending media when they receive an INVITE/SDP message and before responding with SIP provisional or final response. A deny listener will be created to discard early media packets received before provisional SIP response with SDP. Media flows will be created on provisional or final SIP response with SDP and the corresponding deny listeners will be deleted. A call table is used to track calls and their associated media flows.

Other SIP request/response messages, like REGISTER, OPTIONS, SUBSCRIBE, NOTIFY, etc are simply passed through the system.

In ALG mode, "per-client" mode is the only natively supported connection-mode for the peer. All other modes must be handled via iRules. By default, there are no routes attached to the siprouter profile and persistence is disabled. BIG-IP does not insert Via header by default for request messages. The response messages are sent based on the associated "per-client" connection.

The ALG operation mode has two operating contexts, firewall (FW) and source address translation (SNAT). The operation-mode attribute of a SIPRouter profile is used to set a SIP routing instance into ALG mode. The operating context is automatically detected by the source address translation mode of the outgoing connection.

### FW ALG mode

- No address translation
- No subscriber registration tracking
- No separate IP Address support for RTCP flows. Both RTP and RTCP use the same connection IP Address.
- Both TCP and UDP control connections wont terminate when 200OK for BYE or ERROR for INVITE is handled. This option is configurable.
- Calls wont be dropped in case of media flow collision. BIG-IP will attempt to create media channel (RTP/RTCP).
- There could be overlapped calls if the multiple caller/callee has same set of media connection attributes. Thus resulting in partial media for a call, for example only audio or video when one of the connection attribute overlaps with another call.
- Media channels once created will updated solely by the media activity, re-invites won't update the idle timeouts of the existing channels. If the re-invites recreate the new media channels, its idle timeout will be set to its default value as configured.
- For 183 Early Media: The media channels gets established upon receiving 183 for Invite; 200OK following 183 response will not affect the existing call. This statement relies on the assumption that both 183 and 200OK have same set of SDP parameters.

### FW ALG Mode Requirements

- Must allow external access to any address for TCP/UDP port 5060
- There should be a virtual server to receive SIP messages on all vlans that expect SIP requests or responses

### Request Routing

The request messages are forwarded to the destination IP and port. No message headers are modified. For each client, a new connection is established to the destination IP and port.

### Response Routing

For response messages, the MR maintains the association of the per-client connection and the response messages are sent on the associated client-side connection. No headers are modified on the response.

### ALG without Source-address-translation Example

In this example, the default router profile "siprouter-alg" for ALG has no routes attached to it and the operation-mode is "application-level-gateway".

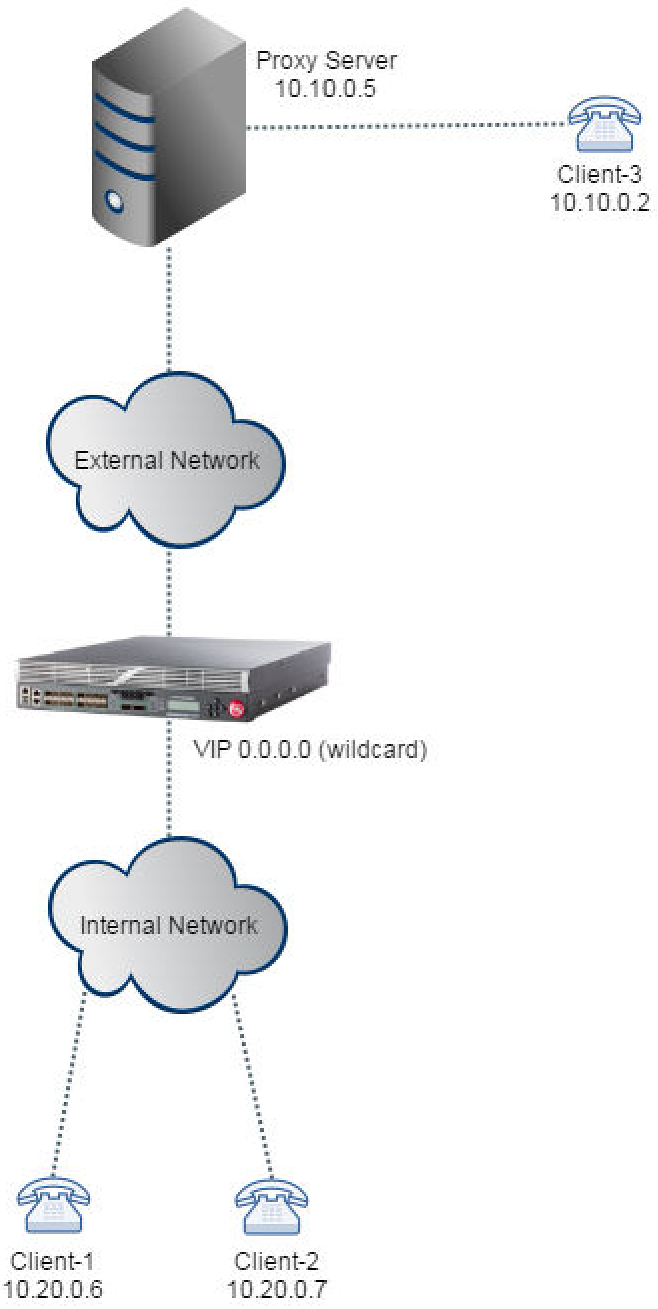


**Figure 11: ALG without source address translation example**

### Configuration

#### Load Balancing Configuration

```
ltm message-routing sip profile router siprouter-alg {
   app-service none
   media-proxy {
       media-inactivity-timeout 120
       max-media-sessions 10
   }
   session {
       max-session-timeout 7200
       transaction-timeout 180
   }
   operation-mode application-level-gateway
}
ltm virtual /Common/vs_sip_alg_udp {
   destination /Common/0.0.0.0:5060
   ip-protocol udp
   mask any
   profiles {
      /Common/sipsession_alg { }
      /Common/siprouter_alg { }
      /Common/udp { }
   }
   source 0.0.0.0/0
   translate-address disabled
   translate-port disabled
}
```

**ALG without Source-address-translation Sequence Diagram**

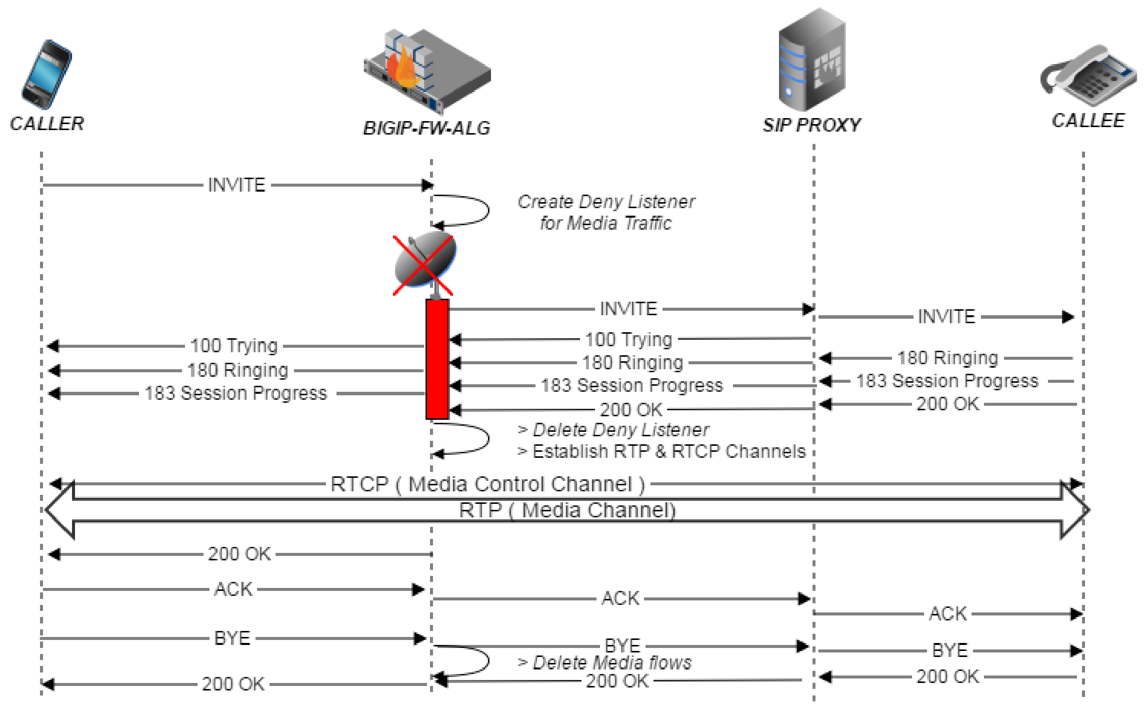


**Figure 12: ALG without source address translation sequence diagram**

# High Availability (HA) Failover

## High Availability (HA) Failover

### Overview

A BIG-IP® system provides high availability via packet mirroring across two chassis. When discussing redundancy, one should consider more than the initial failover. If the backup chassis also fails a fail-back will be required. The following tables provides a quick summary of the initial failover and the fail-back scenarios. Note, a BIG-IP system does not support Geo-Redundant failover. In other words, a BIG-IP system supports the concept of a local HA Pair. However, a BIG-IP system does not support a second HA Pair which will take over if the first HA Pair fails. This type of scenario is required where multiple redundant data centers are available to handle geographic failure scenarios.

#### SIP HA Support

**Table 17: SIP HA Support**

| | Control Messages | Media |
|---|---|---|
| Failover (active failed) | Supported | Supported |
| Failback (new active failed after old active came back online) | Supported | Partial Support |

The above table shows that after the initial failover, if a second failover happens, the media streams, created after the failover, will be maintained. However, media streams created prior to the initial failover will be dropped.

### Configuration Guideline

#### For each BIG-IP in Traffic Group

1. Check the tmm count to be same on each blade as well as each device.
   a. `tmsh list sys db provision.tmmcountactual.`
2. Load the stable build from your branch. (Make sure it's the same build on each device)
3. Load the default config. (Start Fresh)
   a. `tmsh load sys config default.`
4. Configure hostname/users on each device
5. Provision the device with Management – "Small", LTM – "Nominal"
6. Exit wizard by clicking 'Finished' on each device
7. Create vlans (internal/external/HA – advised to create 3 vlans to keep traffic discrete)
   a. GUI:
      a. **Network >> vlans >> new**
   b. TMSH:

   a. `tmsh create net vlan <VLAN_NAME> interfaces add {1.1 {tagged}} tag <TAG_ID>`

8. Create self-ip for each vlan and floating self-ip for both internal and external vlan ( internal,external and HA- with traffic-group-local-only and internal_float, external_float with traffic-group-1)
   a. GUI:
      a. **Network >> self-ips >> new**
   b. TMSH:
      a. `tmsh create net self <SELF_IP_NAME> address <IP_ADDRESS/PREFIX> allow-service default traffic-group <TRAFFIC-GROUP-NAME> vlan <VLAN_NAME>.`
9. Set Config sync address:
   a. GUI:
      a. **Device Management >> Devices >> (self device) >> Device Connectivity >> ConfigSync**
      b. Specify HA self ip
   b. TMSH:
      a. `tmsh modify cm device <DEVICE_NAME> configsync-ip <SYNC_SELF_IP>`
10. Set Mirror address (if mirroring is desired): (For clusters make sure network mirroring is "Between Clusters".)
    a. GUI:
       a. **Device Management >> Devices >> (self device) >> Device Connectivity >> Mirroring**
       b. Specify HA self ip
    b. TMSH:
       a. `tmsh modify cm device <DEVICE_NAME> mirror-ip <MIRROR_SELF_IP>`
11. Set Failover unicast address(es): (GUI preferred)
    a. GUI:
       a. **Device Management >> Devices >> (self device) >> Device Connectivity >> Failover**
       b. Add
       c. Specify HA self ip (as well as mgmt. Ip for backup)
    b. TMSH:
       a. `tmsh modify cm device <DEVICE_NAME> unicast-address { { effective-ip <HA_IP_ADDRESS>} { effective-ip <MGMT_IP_ADDRESS>}}`

## Primary BIG-IP

1. Discover device(s) for trust: (GUI Preferred)
   a. GUI:
      a. **Device Management >> Device Trust >> Peer List**
      b. Add…
      c. Enter IP and credentials for peer device
   b. TMSH:
      a. `tmsh modify cm trust-domain /Common/Root ca-devices add { <IP_OF_REMOTE_DEVICE>} name <NAME_OF_REMOTE_DEVICE> username admin password <ADMIN_PASSWORD>`
2. Each device should now have a trust-sync created device group (not visible) and should show as 'In Sync' and ACTIVE
3. Create a device-group of type sync-failover

   a. GUI:
      a. **Device Management >> Device Groups**
      b. new
      c. Enter name
      d. Specify type of sync-failover
      e. Specify network failover
      f. Add both devices
      g. Save
   b. TMSH:
      a. `tmsh create cm device-group <DGFO_NAME> devices add {<DEVICE_1_NAME>… <DEVICE_2_NAME>} type sync-failover network-failover enabled`
4. Perform initial sync of device-group failover.
   a. GUI:
      a. **Device Management >> Overview >> Select device-group failover >> Select a device**
      b. Click ‘Sync’
   b. TMSH:
      a. `tmsh run cm config-sync to-group <DGFO_NAME>`
5. Devices should now show as ‘In sync’, but one should be ACTIVE the other STANDBY.

# iRule Support

## iRule Support

### Overview

An iRule is a powerful and flexible feature within the BIG-IP® local traffic management system that you can use to manage your network traffic. It allows operators to implement custom behavior beyond the native capabilities of the BIG IP system.

MRF SIP provides a set of iRule events which are raised during message processing and routing which allow operators to inspect and edit the SIP messages. They allow operators to forward, route, reject or drop messages.

Events order for SIP REQUEST message:

CLIENT_DATA (or SERVER_DATA) -> SIP_REQUEST -> MR_INGRESS -> MR_EGRESS -> SIP_REQUST_SEND

Events order for SIP RESPONSE message:

SERVER_DATA (or CLIENT_DATA) -> SIP_RESPONSE -> MR_INGRESS -> MR_EGRESS -> SIP_RESPONSE_SEND

### MRF iRule Events and Commands

#### MRF Events

**Table 18: MRF Events**

| Event | Description |
|---|---|
| MR_INGRESS | This event is raised when a message is received by the message proxy and before a route lookup occurs. Setting the route for a message will bypass route lookup. |
| MR_EGRESS | This event is raised after the route has been selected and processed and the message is delivered to the mr_proxy for forwarding on the new connflow. |
| MR_FAILED | This event is raised when a message has been returned to the originating flow due to a routing failure. |

#### MRF Commands

**Table 19: MRF Events**

| Command | Description |
|---|---|
| MR::instance | Returns the name of the current mr_router instance. The instance name will be the same name as the router profile. |
| MR::protocol | Returns ‘generic, ‘sip’ or ‘diameter’ |
| MR::store <name> … | Stores a tcl variable with the mr_message object. This variable will be delivered with the message to the egress connflow. Adding variables does not effect the content of the message |

| Command | Description |
|---|---|
| MR::restore [<name> …] | Returns adds the stored variables to the current context tcl variable store. If no name is provided, it will add all stored variables. |
| MR::peer <name> | Returns the content of the named peer. If a local peer has been created with the provided name (using MR::peer <name> ...), the local peer's contents will be returned. If a local peer has not been created with the provided name, the static peer from configuration will be returned. The returned value will be formatted as:<br>(versions 11.5 - 12.1)<br><destination> using <transport><br>where:<br>destination = <destination_type> "<destination_value>"<br>destination_type = pool \| virtual<br>transport = <transport_type> "<transport_name>"<br>transport_type = virtual \| config<br>for example:<br>`pool "/Common/default_pool" using config "/Common/ sip_udp_tc"`<br>(version 13.0 + )<br><transport> <destination><br>where:<br>destination = <destination_type> <destination_value><br>destination_type = pool \| virtual<br>transport = <transport_type> <transport_name><br>transport_type = virtual \| config<br>for example:<br>`virtual /Common/sip_tcp_vs host [10.2.3.4]%0:5060` |
| MR::peer <name> [[virtual <virtual_name>] OR [config <transport_config_name>]] [[host <host tuple>] OR [pool <pool name>]] | Defines a peer to use for routing a message to. The peer may either refer to a named pool or a tuple (IP address, port and route domain iD). When creating a connection to a peer, the parameters of either a virtual server or a transport config object will be used. The peer object will only exist in the current connections connflow. When adding a route (via MR::route add), it will first look for a locally created peer object then for a peer object from the configuration. Once the current connection closes, the local peer object will go away. |
| MR::peer <name> [[virtual <virtual_name>] OR [config <transport_config_name>]] [[host <host tuple>] OR [pool <pool name>]] ratio <ratio_value> | Defines a peer to use for routing a message to. The peer may either refer to a named pool or a tuple (IP address, port and route domain iD). When creating a connection to a peer, the parameters of either a virtual server or a transport config object will be used. The peer object will only exist in the current connections connflow. When adding a route (via MR::route add), it will first look for a locally created peer object then for a peer object from the configuration. Once the current connection closes, the local peer object will go away. Adding the ratio keyword allows setting the ratio of the peer. |

| Command | Description |
|---|---|
| MR::message lasthop | Returns the message's lasthop (details of the connection that originated the message). The lasthop is presented as <TMM number>:<FlowID><br><br>for example<br><br>`0:800000000005` |
| MR::message nexthop | Returns the message's nexthop (details of the connection the message is to be forwarded to). If the new_nexthop parameter is present, a nexthop may be set for the message. The nexthop is formated as <TMM number>:<FlowID><br><br>for example<br><br>`0:800000000029` |
| MR::message nexthop <new_nexthop> | Sets the message's nexthop (details of the connection the message is to be forwarded to). The new_nexthop parameter is present, a nexthop may be set for the message. The nexthop is formated as <TMM number>:<FlowID> |
| MR::message route | Returns a rendering of the mr_route_value selected for this message. The returned value will be formatted as:<br><br>(versions 11.5 - 12.1)<br><br>{ <destination> using <transport> [<destination> using <transport>] }<br><br>where:<br><br>destination = <destination_type> "<destination_value>"<br><br>destination_type = pool \| virtual<br><br>transport = <transport_type> "<transport_name>"<br><br>transport_type = virtual \| config<br><br>for example:<br><br>`{ pool "/Common/default_pool" using config "/Common/sip_udp_tc" host "[10.2.3.4]%0:5060" using virtual "/Common/sip_tcp_vs" }`<br><br>(version 13.0 + )<br><br><transport> <destination> [<transport> <destination>]<br><br>where:<br><br>destination = <destination_type> <destination_value><br><br>destination_type = pool \| host<br><br>transport = <transport_type> <transport_name><br><br>transport_type = virtual \| config<br><br>for example:<br><br>`virtual /Common/sip_tcp_vs host [10.2.3.4]%0:5060 config /Common/sip_udp_tc pool /Common/default_pool` |
| MR::message route peer <peer_name> [peer <peer_name>] | Instructs the route table to route the message to the provided peer list. This form of the MR::message route command takes the names of configured peers or dynamic peers created via the MR::peer command. |

| Command | Description |
|---|---|
| MR::message route mode <sequential \| ratio> peer <peer_name> [peer <peer_name>] | Instructs the route table to route the message to the provided peer list. The peer list will have the peer-selection-mode set the the provided mode. This form of the MR::message route command takes the names of configured peers or dynamic peers created via the MR::peer command. |
| MR::message route [[virtual <virtual_name>] OR [config <config_name>]] [[host <host tuple>] OR [pool <pool_name>]] | Instructs the route table to route the message to the provided host or pool. |
| MR::message attempted | Returns a list of hosts that the message has been routed towards. The returned value will be formatted as:<br><br><transport> <destination> [<transport> <destination>]<br><br>where:<br><br>destination = <destination_type> host <host_value><br><br>transport = <transport_type> <transport_name><br><br>transport_type = virtual \| config<br><br>for example:<br><br>`virtual /Common/sip_tcp_vs host [10.2.3.4]%0:5060 config /Common/sip_udp_tc host [20.3.4.5]%0:5060` |
| MR::message attempted none | Clear list of attempted hosts from the message. |
| MR::message attempted [[virtual <virtual_name>] OR [config <config_name>]] [host <host tuple>] | Sets the list of attempted hosts in the message. If set before routing (during MR_INGRESS or MR_FAILED), the hosts in the attempted hosts list will be avoided when performing a lb_pick. |
| MR::message originator | Returns the transport type, transport name and ip address/port/route domain ID of the originator of the message.<br><br>The returned value will be formatted as:<br><br><transport> <destination><br><br>where:<br><br>destination = host <host_value><br><br>transport = <transport_type> <transport_name><br><br>transport_type = virtual \| config<br><br>for example:<br><br>`virtual /Common/sip_tcp_vs host [10.2.3.4]%0:5060` |
| MR::message drop <reason> | Drops the current message |

| Command | Description |
|---|---|
| MR::message retry_count | Returns the number of attempts to route this message that have occurred. |
| MR::message status | Returns the status of the routing operation (valid only at MR_EGRESS). Possible values are: "unprocessed", "route found", "no route found", "dropped", "queue_full", "no connection", "connection closing", "internal error", "persist key in use", and "standby dropped" |
| MR::flow_id | Returns the flow_id of the current connection (in hex). |
| MR::transport | Returns the transport type and name of the current connection.<br>for example<br>`config /Common/sip_udp_tc` |
| MR::prime [config <config_name>] OR [virtual <virtual_name>] [host <host tuple>] OR [pool <pool name>] | Initialize a connection to the specified peer (or active poolmembers of the specified pool) using the specified transport. |
| MR::retry | This command is only available during MR_FAILED event. It re-submits the current message for routing to an alternate pool member. If the previous routing attempt set the message's nexthop or route, these fields should be cleared before retrying routing (use "MR::message nexthop none" and "MR::message route none"). The message's route_status will automatically be reset by this command. If the the retry also fails and the retry_count has reached the max_retries setting in the router profile, the message will be given a "Max retries exceeded" route status. |
| MR::max_retries | Returns the configured max_retries of the router instance. |
| MR::connection_instance | Returns the instance number and number of connections of the current connection within a peer. It will be formatted as "<instance_number> of <max_connections>". For incoming connections, this will return "0 of 1".<br>for example<br>`0 of 5` |
| MR::connection_mode | Returns the connection_mode of the current connection as configured in the peer object. Valid connection_modes are "per-peer, per-blade, per-tmm and per-client". For incoming connections, this will be "per-peer". |

### Route Status

**Table 20: Route Status**

| Status | Description |
|---|---|
| unprocessed | Message has not been submitted for routing yet |
| route found | Route has been found |
| no route found | A route has not been found |
| dropped | The message has been dropped by a MR::message drop command |

| Status | Description |
|---|---|
| queue full | The message was returned back to the originator because one of the MRF processing queues had reached its configured limit. |
| no connection | The message was routed to a connection which was no longer present. |
| connection closing | The message was queued to be send on a connection which was closed. |
| internal error | The message was unable to be delivered due to an internal error. For example, out of memory. |
| persist key in use | Two messages routed using the same persistence key simultanously tried to create the same persistence record. |
| standby dropped | The message is a mirrored message running on a standby device and was dropped as part of routing to avoid creating an outgoing connection on the standby device. |
| Max retries exceeded | The message was returned to the originator because the latest attempt to retry routing exceeded the configured max retry count. |

## SIP iRule Events and Commands

All the SIP/SDP iRule commands specified in the following links are supported.

*https://devcentral.f5.com/wiki/iRules.SIP.ashx*

*https://devcentral.f5.com/wiki/iRules.SDP.ashx*

**Table 21: SIP iRule events and commands**

| Command | Description | Valid SIP Events | Valid MR Events |
|---|---|---|---|
| SIP::persist reset | Deletes any persistence entry with the current persist key of this message. | SIP_REQUEST<br>SIP_RESPONSE<br>SIP_REQUEST_SEND<br>SIP_RESPONSE_SEND | MR_INGRESS<br>MR_EGRESS<br>MR_FAILED |
| SIP::message | Returns the full content of the request or response message. | SIP_REQUEST<br>SIP_RESPONSE<br>SIP_REQUEST_SEND<br>SIP_RESPONSE_SEND | MR_INGRESS<br>MR_EGRESS<br>MR_FAILED |
| SIP::persist [new-persist-key] | Returns the persistence key being used for the current message. If new-persist-key is provided, the existing persistence key is replaced. The value of the new-persist-key MUST be one of valid header value in the message. A header name should not be given as | SIP_REQUEST | MR_INGRESS [For response messages returns EMPTY string] |

| Command | Description | Valid SIP Events | Valid MR Events |
|---|---|---|---|
| | the new-persist-key value. | | |
| SIP::route_status | Returns the routing status of the current message. Valid status are { "unprocessed", "route found", no route found", "dropped", "queue full", "no connection", "connection closing", "internal error" }. "route found" is based on the SIP RouteTable finding a route. It is not effected by the proxy's ability to create a connection, so even if the server is not listening on the specified address or marked down, it might still return status as "route found" if the RouteTable is able to find the route. | SIP_REQUEST_SEND<br>SIP_RESPONSE_SEND | MR_INGRESS<br>MR_EGRESS<br>MR_FAILED |
| SIP::persist replace | Route the message using the route table (or iRule command). On completion of routing, add a new persistence record if one does not exist. I an existing persistence record exists, replace the persistence record with the route selected. | SIP_REQUEST<br>also operates in<br>SIP_RESPONSE<br>SIP_REQUEST_SEND<br>SIP_RESPONSE_SEND | MR_INGRESS<br>MR_FAILED<br>also operates in<br>MR_EGRESS |
| SIP::persist bypass | Route the message using the route table (or iRule command). On completion of routing, add a new persistence record if one does not exist. If an existing persistence record exists, the existing record will not be replaced and the selected route will not be modified. | SIP_REQUEST<br>also operates in<br>SIP_RESPONSE<br>SIP_REQUEST_SEND<br>SIP_RESPONSE_SEND | MR_INGRESS<br>MR_FAILED<br>also operates in<br>MR_EGRESS |
| SIP::persist ignore | Route the message using the route table (or iRule command). The results of the routing will not be | SIP_REQUEST<br>also operates in<br>SIP_RESPONSE | MR_INGRESS<br>MR_FAILED<br>also operates in |

| Command | Description | Valid SIP Events | Valid MR Events |
|---|---|---|---|
| | stored in the persistence table. | SIP_REQUEST_SEND<br>SIP_RESPONSE_SEND | MR_EGRESS |
| SIP::persist [persist-key] [new-timeout-value] | Update the persistence kay and timeout to the new persist-key and new-timeout-value. The persistence key will be used for persistence lookup, add and update. If a persistence value is added or updated, the provided timeout will be used. | SIP_REQUEST<br>also operates in<br>SIP_RESPONSE<br>SIP_REQUEST_SEND<br>SIP_RESPONSE_SEND | MR_INGRESS<br>MR_FAILED<br>also operates in<br>MR_EGRESS |
| SIP::persist use | Use the current persistence record for routing the message if present. If not present, route the message using the route table. On completion of routing, add a new persistence record if one does not exist. If an existing persistence record exists, repleace the message's selected route with the destination stored in the persistence entry. | SIP_REQUEST<br>also operates in<br>SIP_RESPONSE<br>SIP_REQUEST_SEND<br>SIP_RESPONSE_SEND | MR_INGRESS<br>MR_FAILED<br>also operates in<br>MR_EGRESS |

## Persist iRule Example

### Get Persist Key

```
when SIP_REQUEST {
  log local0. "Persist-key = [SIP::persist]"
}
```

### Set Persist Key

```
when SIP_REQUEST {
  SIP::persist [SIP::header value From]
  log local0. "New Persist-key = [SIP::persist]"
}
```

## SIP::header subcommands

Following subcommands are added to SIP::header commands. The values in [square braces] are optional-fields.

**Table 22: SIP::header subcommands**

| Command Name | Description |
|---|---|
| SIP::header count [header-name] | Returns the count of the SIP headers. If "header-name" is specified count the specific headers. |
| SIP::header exists "header-name" | Returns whether SIP header specified by name exists at least once. |
| SIP::header values [header-name] | Returns list of the values of all the instances of SIP header values. If optional argument [header-name] is specified retrieve all values of the specified header-name. |
| SIP::header at "index" | Returns SIP header at "index", index is the Nth line from the SIP header. Returns only the name of the header. |
| SIP::header replace "header-name" "header-value" [index] | Replaces first instance of the header specified by "header-name". New entry is added if not present already. If [index] optional argument is present, replace the header name at [index]th position. |
| SIP::header names | Returns list of all the SIP header names. |

# Troubleshooting

## Troubleshooting

### Log Messages

#### Configuration Validation Errors

**Table 23: Configuration Validation Errors**

| Configuration Failure Condition | Error Message |
|---|---|
| Virtual server config does not have SIP router profile but has SIP session profile | SIP session profile requires SIP router profile also to be assigned to the virtual server <name> |
| Virtual server config does not have SIP session profile but has SIP router profile | SIP router profile requires the SIP session profile to also be assigned to the virtual server <name> |
| Deleting a route which is in use by a SIP router profile | The route <name> is referenced by one or more router profiles |
| Deleting a peer which is in use by a SIP route. | The peer <name> is referenced by one or more SIP routes. |
| Duplicate route attached to router. | Same [request-uri, from-uri, to-uri, virtual-server-name] combination in route <name> exists in the profile <name> |
| Peer refers to non-existent route. | Peer <name> refers to non-existing Static-Route <name> |
| Route refers to non-existent peer. | Static Route <name> refers to non-existing Peer <name> |

#### Connection Termination Reasons

If logging of reset cause is enabled via the tm.rstcause.log db variable, the reason for connection termination is logged to /var/log/ltm.

Reset reason examples:

**Table 24: Connection Termination Reasons**

| Reason Why Text | Description |
|---|---|
| SIP Error | Unexpected internal signaling |
| SIP parser error | Unable to parse SIP message on a stream transport (like TCP) |

#### MRF SIP Troubleshooting Logs

If MRF SIP diagnostic log events are enabled via the log.mrsip.level db variable, the following events will be logged to /var/log/tmm?.

**Table 25: MRF SIP Troubleshooting Logs**

| Event Text | Log Level | Description |
|---|---|---|
| MR SIP: Invalid config attribute <name> in profile <name> | Error | An unexpected configuration attribute was found. For example, an unsupported persist-key was used. |
| MR SIP: Missing header <name> in the message | Error | One of the mandatory SIP header attributes (To, From, Call-ID, Route, Via) was missing. Since the message will not be accepted without the required attributes, this error occurs when an iRule script removes all instances of one of the required scripts after parsing. |
| MR SIP: Decrypt branch parameter failed with error : <error_text> | Error | Unable to decrypt our generated Via header. |
| MR SIP: Encrypt branch parameter failed with error : <error_text> | Error | Unable to encrypt our generated Via header. |
| MR SIP: Generation of AES encryption key failed | Error | Unable to generate AES encryption key for Via header encryption. |
| MR SIP: Parse error reading number for <value> value near <offset> Status code <status code> | Notice | Unable to parse a number for the specified value near the specified offset of an input SIP message. If error response is configured, the specified status code response will be returned and the corresponding stats counter will be incremented. |
| MR SIP: Parse error bad sip protocol version in headline near <offset>. Status Code <status code> | Notice | Invalid sip protocol version near the specified offset of an input SIP message. If error response is configured, the specified status code response will be returned and the corresponding stats counter will be incremented. |
| MR SIP: Parse error invalid or malformed uri in headline near <offset>. Status Code <status code> | Notice | The SIP URI in the headline of an input SIP message is invalid or malformed. If error response is configured, the specified status code response will be returned and the corresponding stats counter will be incremented. |
| MR SIP: Parser error invalid headline near <offset>. Status Code <status code> | Notice | The headline of the incoming sip message is invalid near the specified offset. If error response is configured, the specified status code response will be returned and the corresponding stats counter will be incremented. |
| MR SIP: Parser error to many headers near <offset>. Status Code %d. | Notice | The incoming SIP message contains to many headers to be processed. The header near the specified offset should be the first header that exceeded the limit. If error response is configured, the specified status code response will be returned and the corresponding stats counter will be incremented. |
| MR_SIP: Parser error extraneous header field near <offset> Status code <status code> | Notice | The incoming SIP message contains a extra field in a header near the specified offset. If error response is configured, the specified status code response will be returned and the corresponding stats counter will be incremented. |

| Event Text | Log Level | Description |
|---|---|---|
| MR_SIP: Parser error header to large near <offset>. Status Code <status code>. | Notice | The incoming SIP message has a header line that is too long near the specified offset. If error response is configured, the specified status code response will be returned and the corresponding stats counter will be incremented. |
| MR_SIP: Parser error missing header code <code>. Status Code <status code>. | Notice | The incoming SIP message is missing a required header. The displayed code is a bit-field that can be decoded with access to the internals of the sip parser. If error response is configured, the specified status code response will be returned and the corresponding stats counter will be incremented. |
| MR_SIP: Parser error CSEQ method does not match headline tag <tag> : <tag>. Status Code <status code> | Notice | The incoming SIP message has a mis-match between the headline tag. If error response is configured, the specified status code response will be returned and the corresponding stats counter will be incremented. |
| MR_SIP: Parser max-forwards value has reached zero. Status Code <status code> | Notice | The incoming sip message has been forwarded too many times while being routed, causing the max-forwards value to be decremented to zero. The BIG-IP® system will not process this message. If error response is configured, the specified status code response will be returned and the corresponding stats counter will be incremented. |
| MR_SIP: Server in maintenance mode. Status Code 503 | Notice | The server has been placed in maintenance mode and will not process traffic. If error response is configured, status code 503 response will be returned and the corresponding stats counter will be incremented. |
| MR_SIP: Loop detected. Status code 482. | Notice | The incoming SIP message contains a SIP round loop. If error response is configured, status code 482 response will be returned and the corresponding stats counter will be incremented. |
| MR_SIP: Missing Media Connection attributes. Status Code 488. | Notice | The incoming SIP message is missing required Media Connection Attributes. If error response is configured, status code 488 response will be returned and the corresponding stats counter will be incremented. |
| MR_SIP: Too many media sessions <count> / <count limit>. Error Code <code> | Notice | The number of media sessions in <count> has exceeded the configured <limit count>. If error response is configured, the specified code response will be returned and the corresponding stats counter will be incremented. |

## SIP Troubleshooting Logs

If MRF SIP diagnostic log events are enabled via the log.mrsip.level db variable, the following events will be logged to /var/log/tmm?.

**Table 26: SIP Troubleshooting Logs**

| Event Text | Log Level | Description |
|---|---|---|
| Max Global Registration limit reached | Error | MR SIP: Subscriber registration failed %s, configured max global registration value :%u reached |
| Concurrent Session Per Subscriber limit reached | Error | MR SIP: concurrent session per subscriber limit %u reached, subscriber cannot make calls: %s |
| Non registered subscriber call out | Error | MR SIP: non registered subscriber %s, call dropped. Change SIP session configuration to allow non registered subscriber call out |
| Subscriber registration failed | Error | MR SIP: subscriber %s, unable to register, received non-2xx SIP response" |
| HUDEVT_SA_COOKIE_ PICK event failed | Error | MR SIP: HUDEVT_SA_COOKIE PICKED event error |
| Listener creation failed | Error | MR SIP: Failed to create Listener %K for the subscriber %s |
| Listener deleted | Error | MR SIP: Listener deleted due translation lookup failure %K |

## sipdb Tool

The sipdb tool will be used to display or delete the persistence or media records from session database. The persistence records are created in LB mode when persistence is configured.

The media records are created in ALG mode.

### Usage

```
sipdb [options]
sipdb --persist [--delete] [--router=name] [--key=value] [--type=persistence_type] [--
ipproto=protocol] [-verbose]
sipdb --media [--delete] [--router=name] [-key=call-id]
sipdb --register [-delete] [--router=name] [-key=subscriber uri]
sipdb --help
```

### Options

**Table 27: Options**

| Option | Description |
|---|---|
| --persist<br><br>-s | Indicates persistence mode. This option should be used to display or delete persistence records. This is the default mode.<br><br>Each record displays the persistence type, persistence key, originating ip:port, destination ip:port, protocol and the time remaining.<br><br>The records are grouped by the SIP router profile.<br><br>To delete the persistence record the record key has to be specified. Details are given below in the example section |
| --media<br><br>-m | Indicates media mode. This option should be used to perform operations on media records. |

| Option | Description |
|---|---|
| | The media mode displays the Callid, Origination IP:RTP Port, RTCP Port, Interface name Destination IP: RTP Port, RTCP Port, Interface name.<br><br>In ALG-Translation mode, the output displays the translated address for the subscriber. |
| --register<br><br>-g | Indicates register mode. This option should be used to perform operations on register records.<br><br>The register mode displays the subscriber private address and translated address and the lifetime of the registration. |
| --help<br><br>-h | Displays the help text. |
| --router = name<br><br>-r name | The sip router profile name. This option is used to filter the output matching records for the specified SIP router profile.<br><br>The default partition '/Common' should be specified. For example '/Common/ siprouter'<br><br>The option can be used for both modes i.e. persist and media modes. |
| --key=value<br><br>-k value | Specifies the key for the session record. The option is used to filter the display with the specific key or delete a specific key.<br><br>For persistence mode the key is either a SIP Call-ID, Source Address or Custom value.<br><br>For Media mode the key is SIP Call-ID.<br><br>For register mode the key is the subscriber uri. |
| --delete<br><br>-d | To delete a particular record.<br><br>This option along with the mode and the key details specifies the record to be deleted.<br><br>For persistence mode to delete a record the router name, key, persistence type and ip proto values have to be specified.<br><br>To delete a media entry the router name and SIP Call-ID needs to be specified. |
| --type = value<br><br>-t value | Type of persistence entry.<br><br>The option is applicable when deleting a persistence record.<br><br>Following are the applicable values.<br><br>[C\|c] — For Call-ID id persistence<br>[S\|s] — For Source Address persistence<br>[O\|o] — For Custom type persistence |
| --ipproto = value<br><br>-p value | Either TCP or UDP.<br><br>The option is applicable when deleting a persistence record. |
| --verbose<br><br>-v | This option is applicable in persistence mode.<br><br>Displays the destination transport and pool name in addition to the default display. |

## Examples

### Default Display of Persistence Entries

```
#sipdb
Router: /Common/siprouter       Number of entries: 1
Key                             Originator             Destination           Proto Timeout
---------------------------------------------------------------------------------------------
------------------------------------
C 1-8834@10.10.20.7             10.10.20.2:35462       10.10.10.2:5060       TCP   175

Router: /Common/siprouter_alg      Number of entries: 1
Key                             Originator             Destination           Proto Timeout
---------------------------------------------------------------------------------------------
------------------------------------
C 1-8835@10.10.20.7             10.10.20.2:35462       10.10.10.2:5060       TCP   175
```

### Verbose Display of Persistence Entries

```
# sipdb -v
Router: /Common/siprouter       Number of entries: 1
Key                             Originator             Destination            Proto Timeout
Transport    Pool Name
---------------------------------------------------------------------------------------------
-----------------------------------------
C 1-8872@10.10.20.7             10.10.20.2:56913       10.10.10.2:5060        TCP   175
vs:vs_sip    sip_pool

Router: /Common/siprouter_alg       Number of entries: 1
Key                             Originator             Destination            Proto Timeout
Transport    Pool Name
---------------------------------------------------------------------------------------------
-----------------------------------------
C 1-8874@10.10.20.7             10.10.20.2:56913       10.10.10.2:5060        TCP   175
vs:vs_sip    sip_pool
```

### To filter the above record for a particular SIP router profile name

```
#sipdb --persist –router /Common/siprouter --verbose
#sipdb --persist –router=/Common/siprouter --verbose
#sipdb --persist -r /Common/siprouter

Router: /Common/siprouter       Number of entries: 1
Key                    Originator             Destination            Proto Timeout
Transport  Pool Name
---------------------------------------------------------------------------------------------
-----------------------------------------
C 1-8872@10.10.20.7    10.10.20.2:56913       10.10.10.2:5060        TCP   175
vs:vs_sip  sip_pool
```

### To filter the record for a persistence key

```
#sipdb --persist –key 1-8872@10.10.20.7 --verbose

Router: /Common/siprouter       Number of entries: 1
Key                    Originator             Destination            Proto Timeout
Transport  Pool Name
---------------------------------------------------------------------------------------------
-----------------------------------------
C 1-8872@10.10.20.7    10.10.20.2:56913       10.10.10.2:5060        TCP   175
vs:vs_sip  sip_pool
```

### To delete the above record

```
sipdb persist --delete --key 1-8872@10.10.20.7   --router /Common/siprouter --type C --
ipproto TCP
```

```
Record Successfully deleted
```

### Moving router and/or virtual to different traffic group

The BIG-IP® system does not support changing the traffic group or a router and/or virtual server. MRF stores state that has a different lifetime than a connection in an internal in-memory database (known as session db). This includes persistence tables (SIP LB), call tables (SIP ALG), and registrations tables (SIP ALG), etc. Records stored in session db are auto replicated between the active and standby device. Part of the key for each entry in the session db is the identifier for a traffic-group. If the traffic-group of a virtual and/or router instance is changed all data stored in session db will be orphaned.

### Config changes not loading, or stats don't show up on new router instance

Most changes to config are applied to existing connections. Changes to the set of profiles used by a connection only apply to new connections. Since many message routing protocols use long lived connections, some config changes will not effect existing connections. For example replacing the router profile used by a virtual server will not apply to existing connections. Thus all traffic on existing connections will still be routed through the previous router instance and the stats for that traffic will be included with the previous router instance. To apply the traffic to the new router instance, the existing connections will need to be closed forcing the clients to create new connections.

### iRule changes not loading

Changes to iRule scripts attached to a virtual or transport-config do not change the scripts executing on existing connections. New connections will use the updated scripts. To cause the new script code to be applied, all existing connections (both client side and server side) will need to be closed and new connections created. This may be avoided by moving the business logic of the script to a procedure as follows:

```
ltm rule mylib {
    proc sip_ingress {} {
        if { [SIP::is_request] and [clientside] } {
            # do something
            # change here
        } else {
            # do something else
            # change here
        }
    }
}
ltm rule routing {
    when SIP_INGRESS {
        call mylib::sip_ingress
    }
}
```

### Dropped UDP datagrams

Dropped UDP datagrams have been observed at very low traffic rates (100 calls per second). One cause has been MPI latency. Try making sure the 'scheduler.hsbpollpode.ltm' db var is set to "always". This has been show to reduce the MPI call latency.

## MRF Debugging

### Did the message reach the message router?

There are multiple places where processing can stop or a failure can occur. The stats of the profiles added to the virtual server (or transport-config) should be used to determine if the message reached the message router. From the transport profile's stats (TCP/UDP/SCTP), it can be determined if packets were received by the transport filter. From the protocol profile's stats (sipsession), it can be determined if the received packets were correctly parsed into messages. If an error was found in the message parsing this should be detectable using the protocol's stats.

The message router profile (siprouter) stats should increment with each message received. The result of each messages routing operation should also be represented in the stats.

Why did the message fail routing

The MR_INGRESS event is raised for each message before it enters routing . Once routing is complete either MR_EGRESS or MR_FAILED event is raised. The message metadata can be logged during these events to help debug the results of routing. Some fields and their usage follows:

| Metadata Field | Populated | Purpose |
|---|---|---|
| lasthop | before MR_INGRESS | Contains the TMM and flow_id of the originating connection of the message |
| nexthop | before MR_INGRESS (or during MR_INGRESS) | Selects the destination connection for the message |
| route | after routing (or during MR_INGRESS) | The value of the selected route (peer list). If set during MR_INGRESS, this route will be used instead of performing route lookup |
| originator | before MR_INGRESS | The IP, port and rtdom_id of the originator of the connection. Also contains the transport type and name of the originating connection. |
| status | after routing | The results of the route lookup |
| attempted | after routing (or during MR_INGRESS) | The list of destination hosts attempted. This list of hosts will be treated as marked down when performing peer selection and load balanced pick. |
| retry_count | after routing | The number of times this message has been submitted for routing |

## Why did the message fail routing

The MR_INGRESS event is raised for each message before it enters routing . Once routing is complete either MR_EGRESS or MR_FAILED event is raised. The message metadata can be logged during these events to help debug the results of routing. Some fields and their usage follows:

| Metadata Field | Populated | Purpose |
|---|---|---|
| lasthop | before MR_INGRESS | Contains the TMM and flow_id of the originating connection of the message |
| nexthop | before MR_INGRESS (or during MR_INGRESS) | Selects the destination connection for the message |
| route | after routing (or during MR_INGRESS) | The value of the selected route (peer list). If set during MR_INGRESS, this route will be used instead of performing route lookup |
| originator | before MR_INGRESS | The IP, port and rtdom_id of the originator of the connection. Also contains the transport type and name of the originating connection. |
| status | after routing | The results of the route lookup |

| Metadata Field | Populated | Purpose |
|---|---|---|
| attempted | after routing (or during MR_INGRESS) | The list of destination hosts attempted. This list of hosts will be treated as marked down when performing peer selection and load balanced pick. |
| retry_count | after routing | The number of times this message has been submitted for routing |

#### *MR:route_status: "queue full"*

One reason for MR_FAILED would be when MR:route_status is set to "queue full". This result can happen when the following conditions are met:

1. MRF-SIP profile with TCP for transport.
2. SIP Peer has very few pool members.
3. One of the pool member is down.
4. Burst of SIP Traffic with message size > 2K Bytes.

There are 2 configurable items (Max-Pending-Messages and Max-Pending-Bytes) in the router config to define the queue capacity. If the incoming traffic is high with large messages then the possibility of filling up the queue increases significantly before the connection request timeout occurs on the pool member which is down.

If the message size is larger than 2k then try increasing the Max-Pending-Bytes first. Otherwise, increase Max-Pending-Messages. If neither increase works, then increase both values.

### Messages received on per-client created connections

All messages received on an outgoing connection created using the per-client connection mode, will automatically be forwarded to the connection that received the request which caused the outgoing connection to be created. This includes request messages received on this connection. This is because the connection acts as a direct connection for communication between the original client and the other device. This routing is done be setting the nexthop of all messages received to the last hop of the original request message.

For example:

A SIP INVITE request is received on a connection from 10.10.10.21 to 10.10.20.50. This message gets routed to proxy server 10.20.30.85 using a transport-config that does not configure SNAT and has a connection-mode of per-client. An outgoing connection will be created from 10.10.10.21 to 10.20.30.85. All messages (whether responses of requests) received on the outgoing connection will be automatically routed to the SIP endpoint at 10.10.10.21 using the original incoming connection.

To route a message received on a per-client created connection to another device, the nexthop field will have to be cleared using the MR::message nexthop none command as follows:

```
when MR_INGRESS {
  MR::message nexthop none
  MR::message route config /Common/other_tc host 10.20.30.40:1234
}
```

## Debugging Request Routing

### Overview

SIP Request routing: Request messages are routed via iRule, Persistence or Route Table.

1. An iRule may direct MRF how to route a message during MR_INGRESS. To set the route, use the ‘MR::message route …’ command.
2. A persistence entry using the same persistence key (often call-id) if present will route a message. In MRF persistence entries are bi-directional and remember both SIP devices communicating in the dialog. The persistence table can be accessed via the sipdb tool. The two endpoints in the persistence table are identified as the originator and the destination. The destination of originator versus destination has to do with which direction the original request message that created the persistence entry. If a message arrives that generates the same persistence key, the address of the source of the message will be matched against the destination in the persistence record to determine which direction the message is flowing.
3. If no persistence record is found, the best route table entry for the router is used to select the destination for the route. Attributes of the message are matched against the message to determine which route applies for the current message. MRF SIP route table implementation can match against the message’s request-URI, from-URI, to-URI and originating virtual server. If the message was received on a connection that was initiated by the BIG-IP, the parameters of that connection were likely defined by a transport-config. Messages arriving on a transport-config connection will not match any routes which are filtered by a virtual server.

## Request Routing Debugging

iRules can be used for route debugging. Remember that the iRule needs to be on each transport in the system (virtual servers and transport-configs). MR_INGRESS event runs on the connection that received the request message. MR_EGRESS event runs on the connection that the message is being sent out. MR_FAILED event runs on the connection that received the request message when a message failed to be routed.

SIP iRule commands can be run in the MR iRule events. MRF communicates with the SIP parser to instruct it as to which message is currently used during the MR event.

To know if a message is a request or a response, the following conditional can be used:

```
If {[SIP::response code] eq “”} { # this is a request message
```

During MR_INGRESS, the message’s route can be examined as follows:

```
Log local0. “route [MR::message route]”
```

The transport type and name can be inspected (in v12.0 and later) via an iRule command as follows:

```
Log local0. “transport [MR::transport]”
```

An example script for MR_INGRESS is as follows:

```
when MR_INGRESS {
  log local0. “transport: [MR::transport] flow_id: [MR::flow_id]”
  if {[SIP::reponse code] eq “”} {
    log local0. “request [SIP::method] persist key [SIP::persist] route [MR::message route]”
  } else {
    log local0. “response [SIP::response code] nexthop [MR::message nexthop] route
[MR::message route]"
  }
}
```

After routing has occurred, the messages route field will be populated with the value of the selected route and either MR_EGRESS will be executed or MR_FAILED. If routing succeeded, the route status will be set to “route found” and MR_EGRESS event will be raised on the outgoing connection. If routing failed, the route status will be set and MR_FAILED event will be raised on the incoming connection.

```
when MR_EGRESS {
  log local0. “transport: [MR::transport] flow_id: [MR::flow_id]”
  if {[SIP::reponse code] eq “”} {
    log local0. “request [SIP::method] persist key [SIP::persist] route [MR::message route]”
  } else {
    log local0. “response [SIP::response code] nexthop [MR::message nexthop] route
[MR::message route]"
  }
}

when MR_FAILED {
  log local0. “transport: [MR::transport] flow_id: [MR::flow_id] route status [MR::message
status]”
  if {[SIP::reponse code] eq “”} {
    log local0. “request [SIP::method] persist key [SIP::persist] route [MR::message route]”
  } else {
    log local0. “response [SIP::response code] nexthop [MR::message nexthop] route
[MR::message route]"
  }
}
```

# FAQ

## FAQ

### Advanced-Protocols License

In versions 11.6 and 12.0, MRF SIP virtual servers will not start without an Advanced Protocols license in addition to the LTM license. The license check happens when configuration is loaded, /var/log/ltm file will contain a "MESSAGE ROUTING SIP feature not licensed" line. Since the check only happens when config is loaded, no additional message will be displayed when trying to connect to the virtual server.

Starting with version 12.1, the Advanced Protocol License requirement is no longer required. Only an LTM license is necessary to use SIP.

### Bi-Directional Persistence

Some persistence types, like Call-ID, write bidirectional persistence records. The entry records both SIP devices involved in the call and the transport used to connect to that device. Messages received using the call-id will be matched against the persistence entry to determine which SIP device the message should be forwarded to.

#### Transport Translation

Transport translation is not supported. In other words, a UDP client connection cannot be sent to a TCP peer and vice versa.

#### Connection Recreation

One interesting thing to consider is the snat setting for the virtual server. Lets say that you have two virtual servers inbound_vs and outbound_vs. Each virtual server has a route which uses a corresponding transport config, inbound_tc and outbound_tc. Calls received by the inbound_vs would be routed to connection created using the settings of inbound_tc. The persistence entry for these calls would contain the inbound_vs as the source transport and the inbound_tc as the destination transport.

Likewise calls received by the outbound_vs would be routed to connections created using the setting of outgound_tc. The persistence entry for these calls would contain the outbound_vs as the source transport and outboind_tc as the destination transport. If a call arrives on a connection created via outbound_tc and a valid persistence entry still exists, it would route to a connection using the outbound_vs transport. If no connection is found, it would create a new outbound connection using the outbound_vs's parameters.

Therefore, the virtual server SNAT setting should be that of the VLAN it is on. This is opposite from traditional BIG-IP virtual servers.

Lets say that the inbound_vs listens on the external VLAN and the outbound_tc is for creating connections on the external vlan. The inbound_vs's SNAT settings are what would be used for creating outgoing connections also on the external VLAN. Inbound_vs's snat setting would never be used for creating connection on the internal VLAN.

In this case, the SNAT settings of the inbound_vs should match the SNAT settings of the outbound_tc. Likewise the SNAT settings of the outbound_vs should match the SNAT settings of the inbound_tc.

## Message Retry

When a message fails to route, it will be returned to the originating connflow and MR_FAILED event will be raised. A iRule script will be able to examine the message and resubmit it for routing via the MR::retry command.

There are multiple steps to routing, to understand how MR::retry will work, you will need to understand the steps. To avoid some of these steps or force a different path you may need to modify some of the metadata contained with the message.

Steps of routing:

1. If the message's nexthop attribute is set, the message will be forwarded to the TMM and flowid specified in the nexthop. To avoid this, the message's nexthop should be cleared via 'MR::message nexthop none'.
2. If the message's route attribute is set, the message will skip persistence lookup and route selection and proceed to peer selection (step 5) and lb_pick. Every time route lookup occurs, the message's route attribute is set. To ensure persistence lookup occurs the route attribute should be cleared via 'MR::message route none'.
3. If persistence is enabled on the originating transport, the generated persistence key (via config or iRule) will be used to look for a persistence record. If a persistence record is found, the message will be forwarded to the host specified in the persistence record (step 7). To remove any previous persistence record stored under the message's key use 'SIP::persist reset' or 'DIAMETER::persist reset' iRule command. NOTE: The DIAMETER command is not yet implemented.
4. The protocol specific route table implementation will lookup the best route for the message based on a protocol specific attributes contained in the message. For SIP, it uses the request-uri, to-uri and from-uri of the message. It is also able to match against the virtual server of the originator of the connection. Once a route is found, the message's route attribute is populated with the route.
5. The route found contains a peer list. A peer is selected from the peer list using the peer selection mode.
6. The selected peer may contain a pool and a transport. If a pool exists, it will select the first active pool member that has not already be attempted for this message. If no pool exists, it will forward the message to the local IP and port of the incoming connection.
7. Once a host has been selected, MRF will look to see if an available connection already exists to the host. If an available connection exists, the message will be egressed to the host via that connection. If an available connection does not exist, a new connection will be created and the message will be forwarded through the new connection

### Examples

*Retry the message to a known existing connection:*

```
when MR_FAILED {
 MR::message nexthop 0:010000010111
 MR::retry
}
```

*Retry the message to a pool of alternate servers*

```
When MR_FAILED {
   MR::message nexthop none
   SIP::persist reset
   MR::message route config /Common/BackupTc pool /Common/BackupPool
   MR::retry
 }
```

*Retry the message via the same persistence key*

```
When MR_FAILED {
  MR::message nexthop none
  MR::message route none
  MR::retry
}
```

*To reroute*

```
When MR_FAILED {
  MR::message nexthop none
  MR::message route none
  SIP::persist reset
  MR::retry
 }
```

*To forward to a host*

```
When MR_FAILED {
  MR::message nexthop none
  SIP::persist reset
  MR::message route config /Common/BackupTc host 10.10.10.10:5060
  MR::retry
}
```

## Connection Auto-Initialization

If a peer object has auto-initialization enabled, the BIG-IP® system will automatically create outbound connections to the active pool members in the specified pool using the configuration of the specified transport-config. For auto-initialization to attempt to create a connection, the peer must be included in a route that is attached to a router instance. For each router instance that the peer is contained in, a connection will be initiated. The auto-initialization logic will verify at a configurable interval if the a connection exists between the BIG-IP and the pool members of the pool. If a connection does not exist, it will attempt to reestablish one.

The first auto-intialization attempt will occur at least one auto-initialization-interval delay from when the object is loaded or changed in the TMM.

If the router instance is not included in any virtual servers, connection auto-initialization will not start. Once the router instance has been included in an enabled virtual server, auto-initialization will begin and will remain running for those peers used by routes attached to the router instance even if the router instance is removed from the virtual server.

If a peer with auto-initialization enabled, is used in multiple router instances, a separate connection will be established for each router instance.

The auto-initialization logic will only attempt to create connections to enabled pool members. If the pool member is marked down by an external monitor it will be ignored unless an inband monitor is also attached.

If mirroring is enabled on the router instance, the active device will initialize outgoing connections. The new outgoing connections will be mirrored to the standby device.

## iRules on all transports

With MRF the outgoing connection may not use the same transport as the incoming connection. Incoming connections are defined via virtual servers. Outgoing connections are often defined with transport-configs. If the same iRule script is desired to run on all connections, the script should be defined for all transports.

For example tests assume a simple load balancing configuration with a virtual server (VS_IN) that is part of a router instance with a single default route. This default route contains a single peer that uses a transport-config (TC_OUT) to define the parameters of the outgoing connection. In this setup, a request message would be received on VS_IN. The request message would ingress on a hudchain configured via the settings of the virtual server. As the message was processed, the SIP_REQUEST and MR_INGRESS events would be raised on the iRule scripts attached to the virtual server. The request message would be forwarded to an outgoing connection configured via the setting of the transport-config. As the message egressed through the outgoing connection, the MR_EGRESS and SIP_REQUEST_SEND events would be raised on the iRule scripts attached to the transport-config. When the response message is received by the outgoing connection, the SIP_RESPONSE and MR_INGRESS events would be raised on the iRule script attached to the transport-config. The response will be forwarded to the connection that originated the request and the MR_EGRESS and SIP_RESPONSE_SEND events would be raised on the iRule script attached to the virtual server.

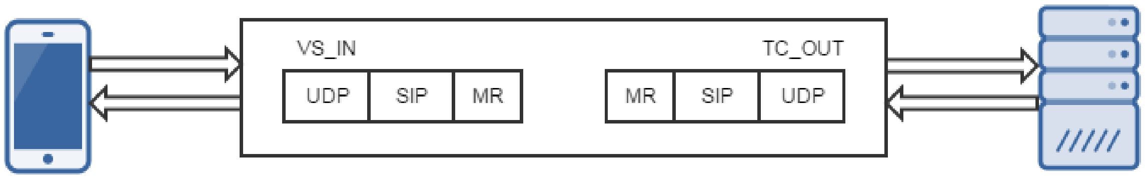


**Figure 13: iRules on all transports**

## Sharing iRule variables between connections

MRF does not join the client side connection with the server side connection (except for SIP ALG). The traditional method of using the CLIENTSIDE or SERVERSIDE keywords to access variables will not work. Instead MRF provides a command to deliver tcl variables along side of the message to the outgoing connection. The MR::store command allows the script author to specify which tcl variables should be delivered to the outgoing connection. The MR::restore command unpacks the delivered variables on the outgoing connection and adds them to the connections context.

For example on the incoming connection:

```
when MR_INGRESS {
  set originator_ip [IP::remote_add]
  set ingress_message_count [expr $message_count + 1]
  MR::store originator_ip ingress_message_count
}
```

On the outgoing connection:

```
when MR_EGRESS {
  MR::restore
  log local0. "originator_ip $originator_ip ingress_message_count $ingress_message_count"
}
```

## The effect of message pipelining on iRule variables

SIP can pipeline messages by allowing messages that require less processing to be forwarded without waiting for earlier messages that require more processing. For this reason, it is not recommended to store state in tcl variables to be used by subsequent iRule events. There is no guarantee that the next event raised after the protocol's message event will be the MR_INGRESS for the same message. For example, saving the SIP uri in a tcl variable during a SIP_REQUEST event to use for making a routing decision during MR_INGRESS is not recommended. The next MR_INGRESS event may not be for the same message as the last SIP_REQUEST event.

MRF SIP implementation allows accessing the SIP iRule commands during the MR events. This is the recommended method to make routing and delivery decisions based on attributes of a message.

For example:

```
when MR_INGRESS {
  if {[URI::host [SIP::uri]] equal "othersp.com"} {
    MR::message route config "/Common/othersp_tc" pool "/Common/othersp_pool"
```

```
   }
}
```

## SNAT settings of the outgoing transport used

MRF uses the SNAT setting of the outgoing connection to determine how the source address is translated. Most outgoing connections are configured via a transport-config and the SNAT setting of the transport config will be used to select the source address. The only time the SNAT settings defined in the virtual server are used is if the setting of the virtual is used to create the outgoing connection (this occurs if no transport-config is set in the peer object).

## Connection Reuse

MRF maintains a table of all existing connections on each TMM of a router instance. When a message is routed to a host, MRF will scan this table for an existing connection to the host that is available for use. If an available existing connection is not found, a new connection will be created.

There are many reasons that an existing connection may not be available for delivery of the current message (see the sub-sections below for details).

### Transport

Each connection is created using the parameters of a transport object (either a virtual server or a transport-config). The transport specifies the profiles, SNAT and iRule scripts of the connection. When a message is routed, MRF will scan the list of connections for a connection created with the same transport specified. Even if the two transports contain the same parameters, a connection created with a different transport will not be used.

A pool object only allows specification of a transport-config as the outgoing connection transport. If the peer object does not specify the transport config, the transport of the message's originating connection will be used. If the system wishes to potentially deliver a message through an existing connection created with by different virtual server on the same router, the MR::message route iRule command must be used. For example:

```
MR::message route virtual "/Common/internal_vs" host [IP::local_addr]:5060
```

### Remote Port and ignore-clientside-port (or ignore-peer-port)

Many clients when creating connections use an ephemeral port for the local port. If a message is routed to that host, the port specified in the host's address will be different than the remote port of any existing connection with the host. Many MRF protocol implementation have an 'ignore_clientside_port' attribute in their router profile. Setting attribute to 'true' instructs MRF that any connection created by the host (client side) that matches the transport, remote IP and rtdom_id may be used.

### Number-connections and instance number

The number-connections attribute of the peer object specifies which connection of a set of connections to a host will be used for delivering a message. It is used alongside the connection-mode instance to set the maximum number of connections between a router instance not the BIG-IP® and a host.

### Connection-mode

Each peer object specifies a connection mode which is used to determine if a connection can be reused or if a new connection should be created. Possible connection modes are:

- per-peer: When a message is routed to a peer with per-peer connection mode, any connection on any TMM with the correct instance number may be used for delivering the message.

- per-blade: When a message is routed to a peer with per-blade connection mode, only connections on the current blade with the correct instance number may be used for delivering the message.
- per-tmm: When a message is routed to a peer with per-tmm connection mode, only connections on the current tmm with the correct instance number may be used for delivering the message.
- per-client: When a message is route to a peer with per-client connection mode, an outgoing connection will be created for exclusive use by the originating connection. The outgoing connection will not be usable for delivering messages from other connections. Any message received (request or response) on the created outgoing connection will be automatically delivered to the originating connection that owns the outgoing connection.

### use-local-connection

Many MRF protocol router profiles contain a 'use-local-connection' attribute. If this attribute is set, if a outgoing connection exists on the current TMM, it will be used even if the instance number does not match. Using this optimization will effectively limit the number of outgoing connections to one per TMM.

### Source port

MRF allows setting the source port used on outgoing connections through the source-port attribute of a transport-config object. Setting this attribute to a non-zero value causes the source port of the outgoing connection to be set to the provided value. If set to zero an ephemeral port value will be used.

Pinning the source port to a fixed value will limit the number of connections available to the host. There can only be one connection using the local and remote tuples (IP/port/rtdom_id) and IP protocol (TCP/UDP/SCTP). Attempts to create another connection using the same addresses and IP protocol will fail.

For this reason it is not recommended to use set the source port for outgoing connections except when using a connection-mode of 'per-peer' and a number-connections of '1'.

Likewise trying to use the same host from peers with different transport settings (transport-config and/or virtual) and setting the source port will produce failures (unless different SNAT pools are used).

## LB Operating Mode

### Response messages being processed by different router instance

SIP routes response messages by inserting a VIA header into the request message. This VIA header contains a branch parameter that is used to contain the internal identifier of the connection that originated the request. The contents of the branch header are encrypted.

When the response message is received, it will contain the inserted VIA header. This inserted header is removed from the message and the branch parameter is decrypted to get the connection identifier of the request's originator. The message will be forwarded to the originating connection. If the originating connection has since been closed, the address in the next top-most header will be used for routing the response message.

This method frees MRF SIP from having to store any data internally while waiting for a response message. All information needed to route the response is added to the message and will be returned with the response. This method works whether the message is returned to the same connection that it was sent from or a new connection.

But if the response message is returned to a different router instance, the branch parameter of the VIA header cannot be decrypted. "MR SIP: Decrypt branch parameter failed with error : Buffer error" will be recorded in /var/log/tmm.

### Response message routing (insert-via and honor-via settings)

SIP can be configured to route responses twice using two different methods.

The first method attempts to route the message to the connection that originated the request message. This is enabled via the 'insert-via' attribute of the sipsession profile. If set, the request message will have a new via header inserted into the message. This via header will identify the IP and port that the next SIP device should route the response to. The response message should contain all the via headers included in the request message. Each SIP device will remove the via header it inserted as the response passes through. An example inserted via is as follows:

```
Via: SIP/2.0/UDP 10.10.10.5:5060;rport;branch=z9hG4bKPjlL6pbh49PLliE2ZNBsASKyO7EBckaoQt
```

When a response is received and insert-via is enabled, the top most via will be removed, and the message will be forwarded to the connection identified by setting the nexthop meta-data field of the message. This can be observed by logging the message's nexthop field during MR_INGRESS event as follows:

```
when MR_INGRESS {
  if {[SIP::response code] ne ""} {
    log local0. "Response: nexthop [MR::message nexthop]"
  }
}
```

If the request's originating connection no longer exists, the MRF proxy will return the message to the connection that received the response. The MR_FAILED event will be raised. Upon completion of the MR_FAILED logic, the message will be returned the the SIP filter. The SIP filter will use the fallback response routing mechanism if the 'honor-via' attribute is enabled. The fallback response routing used the IP and port of the second topmost VIA header of the received response (now the topmost after deleting the inserted one). This is the via header that was topmost when the request message was received. This header should contain the IP and port that the device which sent the request to the BIG-IP.

The MRF SIP filter will clear the message's nexthop field and instead set the message's route meta-data field to route the message to the IP and port of the device which sent the request to the BIG-IP. Once the route field has been set, the message will again be forwarded to the MRF proxy for routing and MR_INGRESS event will be raised.

***Note:*** *The route command will specify the transport of the connection that received the response as the transport to use when creating the connection to the source of the request. If this is not desired, the route field can be modified during the subsequent MR_INGRESS event.*

## ALG without SNAT (No Address Translation)

The Secure Real-time Transport Protocol (SRTP)- RFC3711 is not supported in this mode.

## Routing using a virtual with SNAT none may select a source port of zero

MRF allows routing to a peer without a transport-config selected. If a peer does not have a transport-config, the transport of the message originating connection will be used to create the outgoing connection. If originating connection used a virtual server as its transport, the serverside of the virtual server will be used to create the outgoing connection.

If the virtual server had a SNAT setting of none and the 'source-port' attribute set to 'preserve' or 'preserve-strict', the outgoing connection will be created with a source port of zero instead of the remote port of the originating connection.

## SIP ENUM Resolution Capability using iRule

DNS team has developed an iRule called RESOLV::lookup to perform a DNS query . From release 11.5.0, its capability was improved to support resolving NAPTR and SRV addresses as well. SIP ENUM resolution mainly involves resolving Telephone number to an IP address. This process normally involves 4 steps namely

- Normalizing the telephone number to an ENUM address format.
- Perform NAPTR resolution on ENUM to retrieve SRV records.
- Perform SRV resolution to retrieve Domain Name records.
- Perform DNS query to retrieve the IP Address.

RESOLV::lookup with its new capabilities could potentially be used to resolve ENUM to IP Address.

Eg: NAPTR resolution to retrieve SRV records.

```
RESOLV::lookup @$static::dns_vs inet -naptr "4.4.2.2.3.3.5.6.8.1.4.4.e164.arpa"
```

# Legal Notices

## Legal notices

### Publication Date

This document was published on June 15, 2018.

### Publication Number

MAN-0670-00

residential area is likely to cause harmful interference, in which case the user, at his own expense, will be required to take whatever measures may be required to correct the interference.

Any modifications to this device, unless expressly approved by the manufacturer, can void the user's authority to operate this equipment under part 15 of the FCC rules.

## Canadian Regulatory Compliance

This Class A digital apparatus complies with Canadian ICES-003.

## Standards Compliance

This product conforms to the IEC, European Union, ANSI/UL and Canadian CSA standards applicable to Information Technology products at the time of manufacture.

# Index