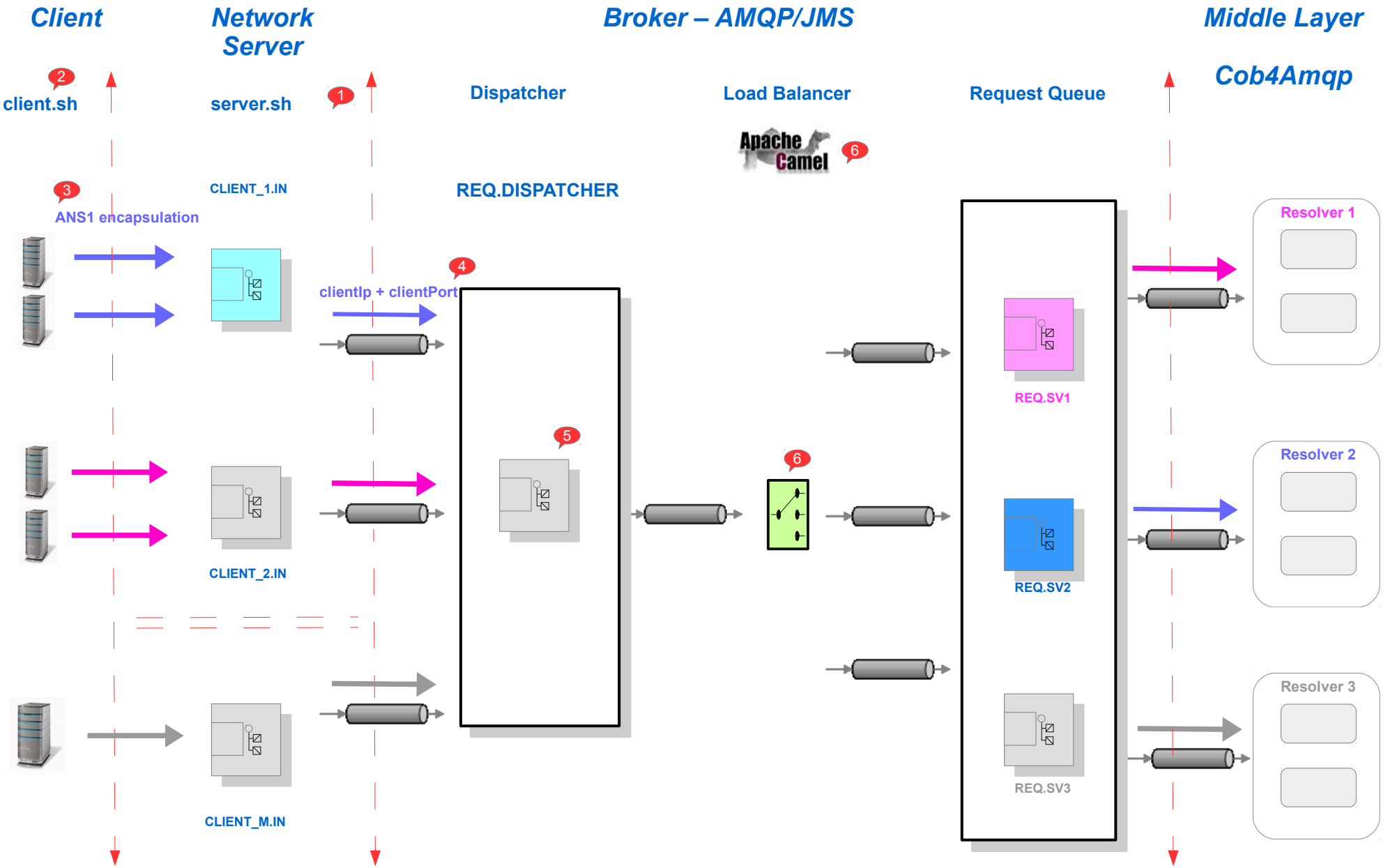
High Performance Messaging System

Processing Request from Client to Broker



Processing Request from Client to Broker

1. A Network server (just one simple server or many, in the same host, or in multiple machines)

    See the configuration files (server_properties.xml and broker_properties.xml)

    Client List, that are allowed to connect to the server in server_properties.xml.

    server.sh starts the server, once deployed.

2. A client opens a connection to the network server that creates 2 threads to proccess sockect input/output asynchronously.

    Every Client (IP) has an inputQueue (CLIENT_i.IN) /outputQueue (CLIENT_i.OUT) in the Broker

    Same IP are able to connect multiple times, and share the same inputQueue/outputQueue

3. A client send a message to the server (ASN.1 encapsulated).

4. The server receives it, unpack the message (as text or binary), create some jms-message properties (clientIp+clientPort+messageUUID) and enqueue and AMQP message to the client's inputQueue (with the previous properties).

    These properties are propagated in all the life cycle for the message.

5. All the input message (from all connected clients) are concentrated in the REQ.DISPATHER queue automatically by ActiveMQ.

6. A Camel Round-Robin routing rule in ActiveMQ moves REQ.DISPATHER messages to a pool of request input queues (REQ.SV1, REQ.SV2, REQ.SV3 ..)

    Every request input queue is assigned to a Resolver Node. (REQ.SVk means requests to resolver k-th Node)

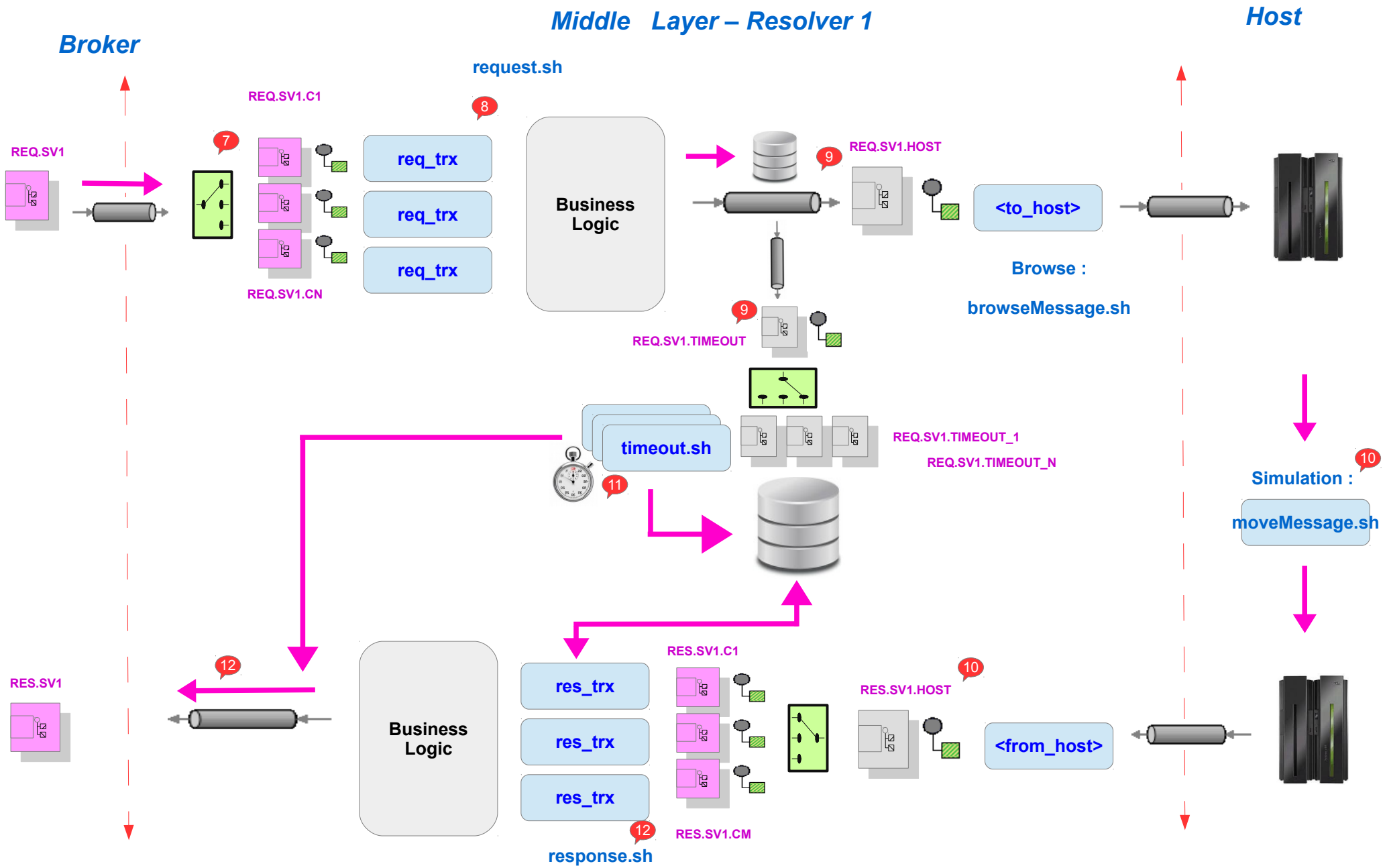
    Multiple input queues/resolvers can be made available for high load

Processing Request from Client to Broker

6. (continues) In our case the following Camel route moves messages from REQ.DISPATCHER to the resolver resquest queue (REQ.SV1, REQ.SV2) in round-robn mode

```
<route>

<!--        Broker Layer - Dispatcher to Resolver's Queues

REQ.DISPATCHER : input queue for dispatching request

REQ.SV1        : request input queue for Resolver Node1

REQ.SV2        : request input queue for Resolver Node2

REQ.SV3        : request input queue for Resolver Node3

-->

            <from uri="activemq:queue:REQ.DISPATCHER"/>

            <loadBalance>

            <roundRobin/>

            <to uri="activemq:queue:REQ.SV1"/>

            <to uri="activemq:queue:REQ.SV2"/>

            <!-- <to uri="activemq:queue:REQ.SV3"/> -->

            </loadBalance>

            </route>
```

Cobol 4 AMQP
Broker to Resolver – Request & Response



Broker to Resolver – Request & Response

7. Every message in REQ.SVi is automatically routed by a Round-Robin routing rule to multiple input queues (REQ.SVi.C1,..,REQ.SVi.CN) in a specific Resolver

8. Messages in REQ.SVi.Ci queue are processed by a server process (request.sh).

    - request.sh implements the business logic for every message request.

9. Every processed message is sent to the Host :

    The message is written to REQ.SVi.HOST

    The message is written to REQ.SVi.TIMEOUT queue and a timeout request is written to DB (a new row in TIMEOUT DB table for the messageUUID).

10.Response Process is simulated by moveMessage.sh :

    Every response is delayed between 1000 to 3000 milliseconds (could be parameretized)

11.Timeout Process is performed by timeout.sh (TimeoutMessage) :

    Listen for messages REQ.SVi.TIMEOUT (inputqueue) , with an entry in TIMEOUT DB table :

    When the timeout is finished an attempt to delete the row from TIMEOUT is executed for the messageUUID.

    In case the delete returns 0 row , the message has been answered from the HOST, and nothing else is necessary.

    In case the delete returns 1 row , the message has NOT been answered from the HOST (which means that has been TIMEOUT), and a TIMEOUT response is generated to RES.SVi (outputqueue)

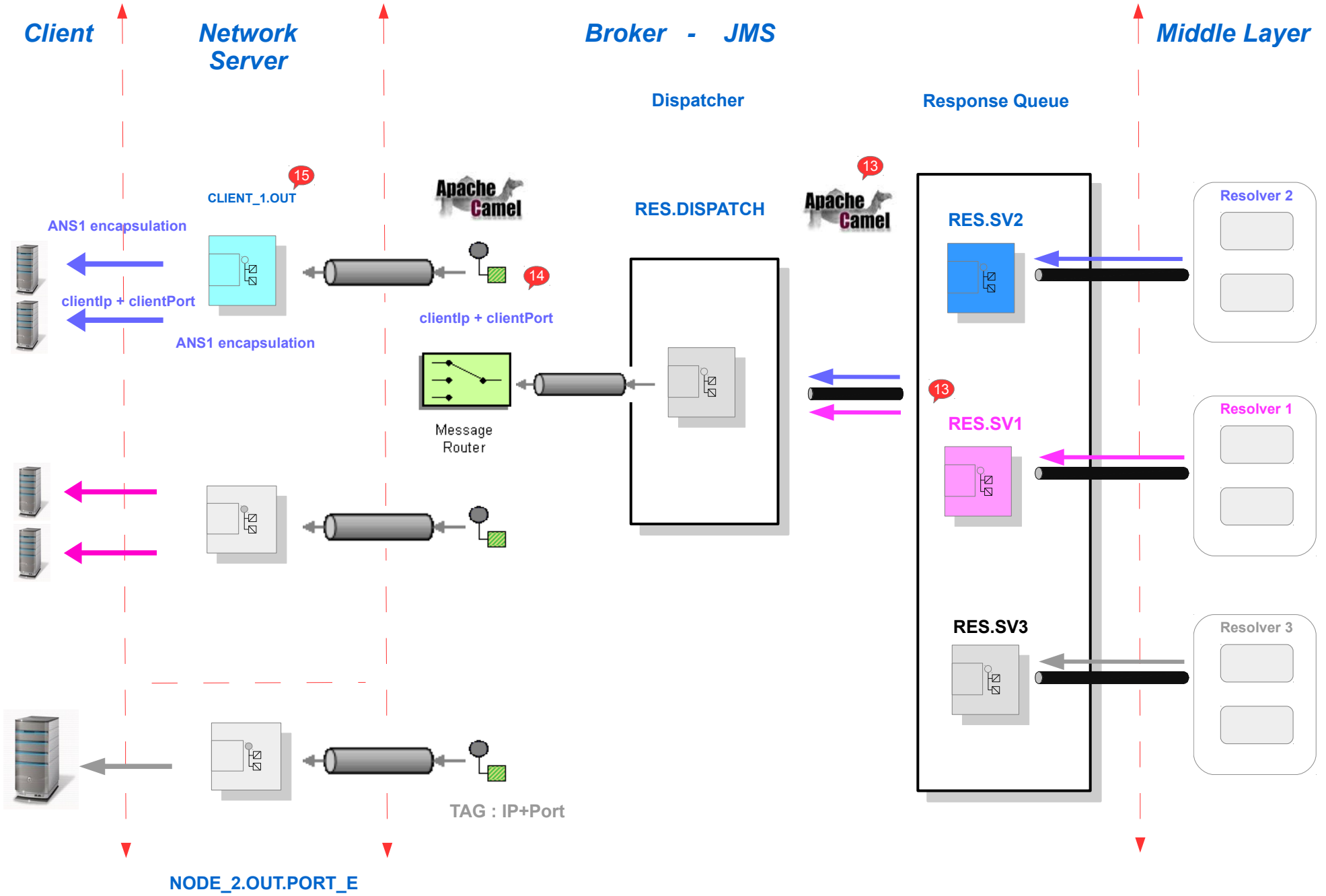
12. Response Process is performed by response.sh:

    Listen for messages RES.SVi.Ci

    deletes the row from TIMEOUT DB Table

    implements the business logic for every message and writes a response to RES.SVi

Returning Response to the Client



Returning Response to the Client

13. Responses from RES.SVi nodes are collected by a Camel rule and written to RES.DISPATH queue

14. A Camel message router moves every message from RES.DISPATH to the corresponding destination

    Inspect the message header an select messages by ${header.clientIp} contains source IP

    The selected message is written to outputQueue (CLIENT_i.OUT) for the socket

15. The server socket for the destination IP:

    Select messages for his own remote port

    Enscapsulates the response as ANS 1 , and send the response

PgBouncer Connection Pooling