Load Balancer

Table of Contents

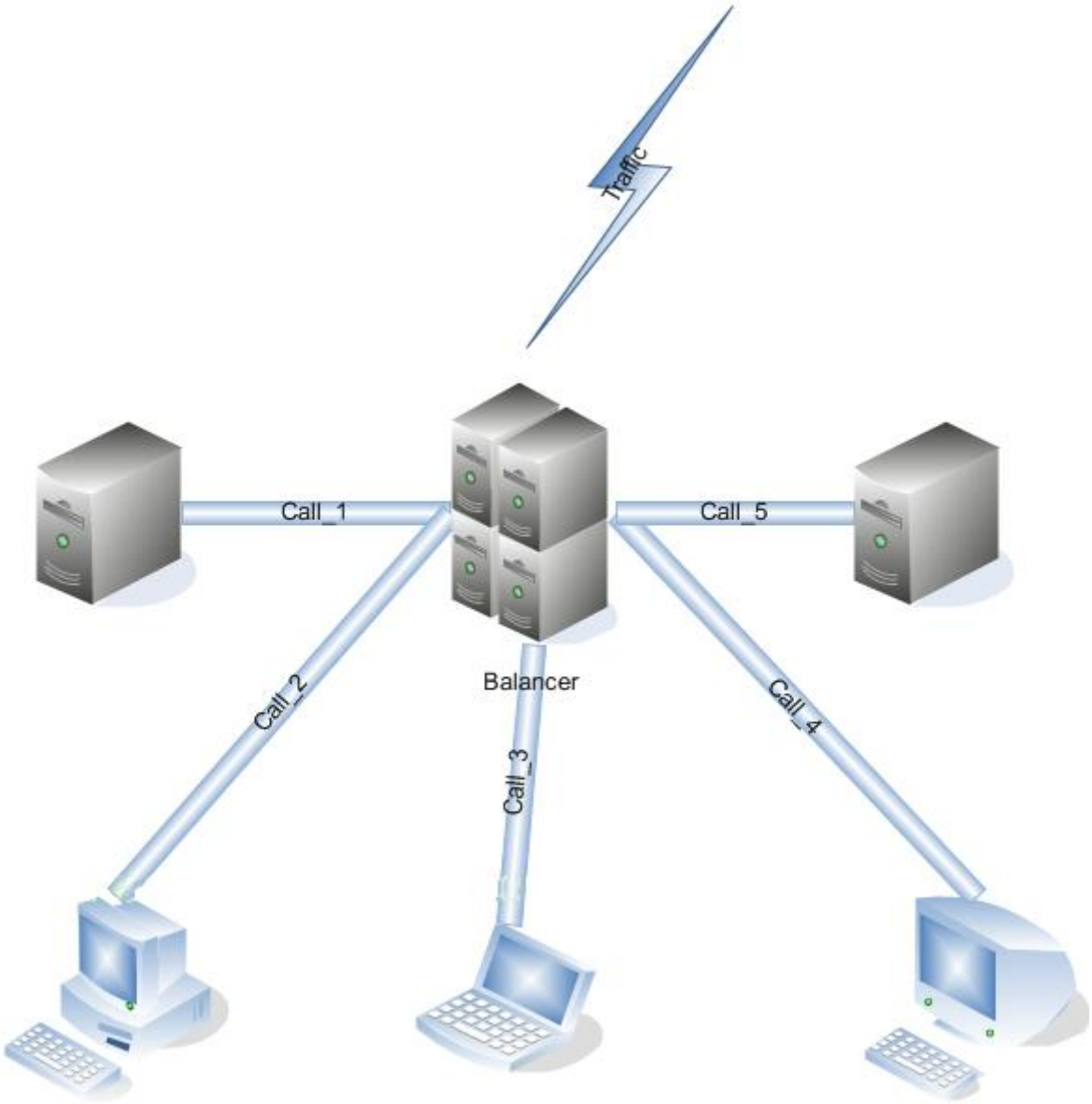

Figure 1. Star Cluster Topology.

The Restcomm Load Balancer is used to balance the load of SIP service requests and responses between nodes in a SIP Server cluster, such as Restcomm JAIN SLEE or SIP Servlets. Both Restcomm servers can be used in conjunction with the Restcomm Load Balancer to increase the performance and availability of SIP services and applications.

In terms of functionality, the Restcomm Load Balancer is a simple stateless proxy server that intelligently forwards SIP session requests and responses between User Agents (UAs) on a Wide Area Network (WAN), and SIP Server nodes, which are almost always located on a Local Area Network (LAN). All SIP requests and responses pass through the Restcomm Load Balancer.

Starting with the 7.0.0.GA release, Restcomm Load Balancer can handle WebSocket requests supporting up to version 13 of the wire protocol - RFC 6455

(version 17 of the draft hybi specification - http://tools.ietf.org/html/draft-ietf-hybi-thewebsocketprotocol-17). Restcomm Load Balancer will accept HTTP requests and if the request contains header Sec-WebSocket-Protocol it will upgrade the connection and dispatch the reuqest to a node capable to handle WebSocket requests (a node that has a WebSocket connector).

One major advantage of having WebSocket support for Restcomm Load Balancer is to allow tunneling over HTTP port for SIP traffic and thus bypass proxy and firewall servers that might be blocking SIP traffic.

SIP and SMPP Load Balancers: Installing, Configuring andRunning

Pre-Install Requirements and Prerequisites

Software Prerequisites

A JAIN SIP HA-enabled application server such as Restcomm JAIN SLEE or Restcomm SIP Servlets is required.

Running the Restcomm Load Balancer requires at least two instances of the application server as cluster nodes nodes. Therefore, before configuring the Restcomm Load Balancer, we should make sure we've installed a the SIP application server first. The Restcomm Restcomm Load Balancer will work with a SIP Servlets-enabled JBoss Application Server *or* a JAIN SLEE application server with SIP RA.

Downloading

The load balancer is located in the *sip-balancer* top-level directory of the Restcomm distribution. You will find the following files in the directory:

SIP and SMPP load balancer executable JAR file

This is the binary file with all dependencies, include SMPP load balancer

SIP and SMPP load balancer Configuration Properties file

This is the properties files with various settings

Installing

The SIP and SMPP load balancer executable JAR file can be extracted anywhere in the file system. It is recommended that the file is placed in the directory containing other JAR executables, so it can be easily located in the future.

Configuring

Configuring the Restcomm Load Balancer and the two SIP Servlets-enabled Server nodes is described in Procedure: Configuring the Restcomm Restcomm Load Balancer and SIPServer Nodes .

Procedure: Configuring the Restcomm Restcomm Load Balancer and SIPServer Nodes

1. Configure lb.properties Configuration Properties File

   Configure the SIP and SMPP Load Balancer's Configuration Properties file by substituting valid values for your personal setup. Complete Sample lb.properties File shows a sample *lb.properties* file, with key element descriptions provided after the example. The lines beginning with the pound sign are comments.

   Example 1. Complete Sample lb.properties File

```
# Mobicents Load Balancer Settings
# For an overview of the Mobicents Load Balancer visit
http://docs.google.com/present/view?id=dc5jp5vx_89cxdvtxcm

# The binding address of the load balancer
host=192.168.1.70

# The RMI port used for heartbeat signals
rmiRegistryPort=2000
# The port to be used used for the Remote Object
rmiRemoteObjectPort=2001

# The SIP port used where client should connect
externalPort=5060
# The SIP TLS port used where clients should connect
externalSecurePort=5061

# The SIP port from where servers will receive messages
# delete if you want to use only one port for both inbound and outbound)
# if you like to activate the integrated HTTP load balancer, this is the entry
point
internalPort=5065
# The SIP TLS port from where servers will receive messages
internalSecurePort=5066

# The HTTP port for HTTP forwarding
httpPort=2080
# The HTTPS port for HTTPS forwarding
httpsPort=2081
#If no nodes are active the LB can redirect the traffic to the unavailableHost
specified in this property,
#otherwise, it will return 503 Service Unavailable
#unavailableHost=google.com
```

```
#Specify UDP or TCP or TLS (for now both must be the same)
internalTransport=UDP
externalTransport=UDP
#Enables ws/wss transport
#isTransportWs = false
# If you are using IP load balancer, put the IP address and port here, for TLS
put externalSecureIpLoadBalancerPort
#externalIpLoadBalancerAddress=127.0.0.1
#externalIpLoadBalancerPort=111
#externalSecureIpLoadBalancerPort=112

# Requests initited from the App Servers can route to this address (if you are
using 2 IP load balancers for bidirectional SIP LB)
# For TLS put internalSecureIpLoadBalancerPort
#internalIpLoadBalancerAddress=127.0.0.1
#internalIpLoadBalancerPort=111
#internalSecureIpLoadBalancerPort=112

# Designate extra IP addresses as serer nodes
#extraServerNodes=222.221.21.12:21,45.6.6.7:9003,33.5.6.7,33.9.9.2

# Call-ID affinity algortihm settings. This algorithm is the default. No need
to uncomment it.
#algorithmClass=org.mobicents.tools.sip.balancer.CallIDAffinityBalancerAlgorith
m
algorithmClass=org.mobicents.tools.telestaxproxy.TelestaxProxyAlgorithm
# This property specifies how much time to keep an association before being
evitcted.
# It is needed to avoid memory leaks on dead calls. The time is in seconds.
#callIdAffinityMaxTimeInCache=500

# Uncomment to enable the consistent hash based on Call-ID algorithm.
#algorithmClass=org.mobicents.tools.sip.balancer.HeaderConsistentHashBalancerAl
gorithm
# This property is not required, it defaults to Call-ID if not set, cna be
"from.user" or "to.user" when you want the SIP URI username
#sipHeaderAffinityKey=Call-ID
#specify the GET HTTP parameter to be used as hash key
#httpAffinityKey=appsession

# Uncomment to enable the persistent consistent hash based on Call-ID
algorithm.
#algorithmClass=org.mobicents.tools.sip.balancer.PersistentConsistentHashBalanc
erAlgorithm
# This property is not required, it defaults to Call-ID if not set
#sipHeaderAffinityKey=Call-ID
#specify the GET HTTP parameter to be used as hash key
#httpAffinityKey=appsession

#This is the JBoss Cache 3.1 configuration file (with jgroups), if not
```

```
specified it will use default
#persistentConsistentHashCacheConfiguration=/home/config.xml

# Call-ID affinity algortihm settings. This algorithm is the default. No need
to uncomment it.
#algorithmClass=org.mobicents.tools.sip.balancer.CallIDAffinityBalancerAlgorith
m
# This property specifies how much time to keep an association before being
evitcted.
# It is needed to avoid memory leaks on dead calls. The time is in seconds.
#callIdAffinityMaxTimeInCache=500

# Uncomment to enable the consistent hash based on Call-ID algorithm.
#algorithmClass=org.mobicents.tools.sip.balancer.HeaderConsistentHashBalancerAl
gorithm
# This property is not required, it defaults to Call-ID if not set, cna be
"from.user" or "to.user" when you want the SIP URI username
#sipHeaderAffinityKey=Call-ID
#specify the GET HTTP parameter to be used as hash key
#httpAffinityKey=appsession

# Uncomment to enable the persistent consistent hash based on Call-ID
algorithm.
#algorithmClass=org.mobicents.tools.sip.balancer.PersistentConsistentHashBalanc
erAlgorithm
# This property is not required, it defaults to Call-ID if not set
#sipHeaderAffinityKey=Call-ID
#specify the GET HTTP parameter to be used as hash key
#httpAffinityKey=appsession

#This is the JBoss Cache 3.1 configuration file (with jgroups), if not
specified it will use default
#persistentConsistentHashCacheConfiguration=/home/config.xml

#JSIP stack configuration.....
javax.sip.STACK_NAME = SipBalancerForwarder
javax.sip.AUTOMATIC_DIALOG_SUPPORT = off
# You need 16 for logging traces. 32 for debug + traces.
# Your code will limp at 32 but it is best for debugging.
# LOG4J means the level will be configurable from the JOG4J config file
gov.nist.javax.sip.TRACE_LEVEL = LOG4J

#Specify if message contents should be logged.
gov.nist.javax.sip.LOG_MESSAGE_CONTENT=false

gov.nist.javax.sip.DEBUG_LOG = logs/sipbalancerforwarderdebug.txt
gov.nist.javax.sip.SERVER_LOG = logs/sipbalancerforwarder.xml
gov.nist.javax.sip.THREAD_POOL_SIZE = 64
gov.nist.javax.sip.REENTRANT_LISTENER = true
gov.nist.javax.sip.AGGRESSIVE_CLEANUP=true
#this property we set statically
```

```
"gov.nist.javax.sip.stack.LoadBalancerNioMessageProcessorFactory"
#for statistic correct working
#gov.nist.javax.sip.MESSAGE_PROCESSOR_FACTORY=gov.nist.javax.sip.stack.NioMessa
geProcessorFactory

#If a node doesnt check in within that time (in ms), it is considered dead.
nodeTimeout=8400
#The consistency of the above condition is checked every heartbeatInterval
milliseconds
heartbeatInterval=150

#SMPP Load Balancer Settings
smppName = SMPP Load Balancer
# The address of the load balancer
smppHost = 127.0.0.1
# The port of the load balancer
smppPort = 2776
# The port of the load balancer for SSL protocol
smppSslPort = 2876
# Remote SMPP servers (use comma between servers and colon between IP and port)
remoteServers = 127.0.0.1:10021,127.0.0.1:10022,127.0.0.1:10023
# it is recommended that at any time there were no more than
#10 (ten) SMPP messages are outstanding
maxConnectionSize = 10
# Is NIO enabled
nonBlockingSocketsEnabled = true
# Is default session counters enabled
defaultSessionCountersEnabled = true
# Response timeout for load balancer in milliseconds
timeoutResponse = 10000
# Session initialization timer
timeoutConnection = 1000
# Enquire Link Timer
timeoutEnquire = 50000
# Time between reconnection
reconnectPeriod = 1000
# Connection check timer in load balancer
timeoutConnectionCheckClientSide = 1000
# Connection check server side timer
timeoutConnectionCheckServerSide = 1000

# SSL configuration
#points to the keystore file we generated before
javax.net.ssl.keyStore=/home/konstantinnosach/tmp/keystore
#provides the password we used when we generated the keystore
javax.net.ssl.keyStorePassword=123456
#points to the truststore file we generated before
javax.net.ssl.trustStore=/home/konstantinnosach/tmp/keystore
#provides the password we used when we generated the truststore
javax.net.ssl.trustStorePassword=123456
#this is important because sipp supports only TLSv1 and so we need to restrict
```

```
the protocols only to that
gov.nist.javax.sip.TLS_CLIENT_PROTOCOLS=TLSv1
#can be : Enabled, Disabled, DisabledAll, Want
#if Enabled, used to request and require client certificate authentication: the
connection will terminate if no suitable client certificate is presented
#if Want, used to request client certificate authentication, but keep the
connection if no authentication is provided
#if Disabled or DisabledAll does not use authentication
gov.nist.javax.sip.TLS_CLIENT_AUTH_TYPE=Disabled
#ssl will provide some extra debugging information for the SSL if uncomment it
#javax.net.debug=ssl
# should smpp and http use secured connection towards servers or regular
isRemoteServerSsl = true

#Statistic
#port for statistic
statisticPort=2006
```

host

Local IP address, or interface, on which the Restcomm Load Balancer will listen for incoming requests.

externalPort

Port on which the Restcomm Load Balancer listens for incoming requests from SIP User Agents.

externalSecurePort

TLS port on which the Restcomm Load Balancer listens for incoming requests from SIP User Agents.

internalPort

Port on which the Restcomm Load Balancer forwards incoming requests to available, and healthy, SIP Server cluster nodes.

internalSecurePort

TLS port on which the Restcomm Load Balancer forwards incoming requests to available, and healthy, SIP Server cluster nodes.

rmiRegistryPort

Port on which the Restcomm Load Balancer will establish the RMI heartbeat connection to the application servers. When this connection fails or a disconnection instruction is received, an application server node is removed and handling of requests continues without it by redirecting the load to the lie nodes.

rmiRemoteObjectPort

Port on which the Restcomm Load Balancer will use to export the RMI Remote Object.

httpPort

Port on which the Restcomm Load Balancer will accept HTTP requests to be distributed across the nodes.

httpsPort

Port on which the Restcomm Load Balancer will accept HTTPS requests to be distributed across the nodes.

internalTransport

Transport protocol for the internal SIP connections associated with the internal SIP port of the load balancer. Possible choices are `UDP` , `TCP` and `TLS` .

externalTransport

Transport protocol for the external SIP connections associated with the external SIP port of the load balancer. Possible choices are `UDP` , `TCP` and `TLS` . It must match the transport of the internal port.

externalIpLoadBalancerAddress

Address of the IP load balancer (if any) used for incoming requests to be distributed in the direction of the application server nodes. This address may be used by the Restcomm Load Balancer to be put in SIP headers where the external address of the Restcomm Load Balancer is needed.

externalIpLoadBalancerPort

The port of the external IP load balancer. Any messages arriving at this port should be distributed across the external SIP ports of a set of Restcomm Load Balancers.

externalSecureIpLoadBalancerPort

The TLS port of the external IP load balancer. Any messages arriving at this port should be distributed across the external SIP ports of a set of Restcomm Load Balancers.

internalIpLoadBalancerAddresst

Address of the IP load balancer (if any) used for outgoing requests (requests initiated from the servers) to be distributed in the direction of the clients. This address may be used by the Restcomm Load Balancer to be put in SIP headers where the internal address of the Restcomm Load Balancer is needed.

internalIpLoadBalancerPort

The port of the internal IP load balancer. Any messages arriving at this port should be distributed across the internal SIP ports of a set of Restcomm Load Balancers.

internalSecureIpLoadBalancerPort

The TLS port of the internal IP load balancer. Any messages arriving at this port should be distributed across the internal SIP ports of a set of Restcomm Load Balancers.

extraServerNodes

Comma-separated list of hosts that are server nodes. You can put here alternative names of the application servers here and they will be recognized. Names are important, because they

might be used for direction-analysis. Requests coming from these server will go in the direction of the clients and will not be routed back to the cluster.

algorithmClass

The fully-qualified Java class name of the balancing algorithm to be used. There are three algorithms to choose from and you can write your own to implement more complex routing behaviour. Refer to the sample configuration file for details about the available options for each algorithm. Each algorithm can have algorithm-specific properties for fine-grained configuration.

nodeTimeout

In milliseonds. Default value is 5100. If a server node doesnt check in within this time (in ms), it is considered dead.

heartbeatInterval

In milliseconds. Default value is 150 milliseonds. The hearbeat interval must be much smaller than the interval specified in the JAIN SIP property on the server machines - `org.mobicents.ha.javax.sip.HEARTBEAT_INTERVAL`

smppName

Name of SMPP load balancer.

smppHost

Local IP address on which the SMPP load balancer will listen for incoming requests from clients.

smppPort

Port on which the SMPP load balancer will listen for incoming requests from clients.

remoteServers

IP adresses and ports of remote SMPP servers.

maxConnectionSize

It is recommended that at any time there were no more than ten SMPP messages are outstanding.

nonBlockingSocketsEnabled

Turn off/on blocking.

defaultSessionCountersEnabled

Turn off/on server counters.

timeoutResponse

In milliseconds. Max time allowable between request and response, after which operation assumed to have failed.

timeoutConnection

In milliseconds. Max time allowable between connection to SMPP load balancer by client and

bind request from client, after which client will disconnect.

timeoutEnquire

In milliseconds. Time interval `after which balancer check connection to server and client.`

reconnectPeriod

In milliseconds. Time period after which balancer reconnects to server if connection to server was lost.

timeoutConnectionCheckClientSide

In milliseconds. After sending enquire link to client for checking connection, balancer wait this time and if not receive response close connection.

timeoutConnectionCheckServerSide

In milliseconds. After sending enquire link to server for checking connection, balancer wait this time and if not receive response close connection.

javax.net.ssl.keyStore

Points to the keystore file we generated before.

javax.net.ssl.keyStorePassword

Provides the password we used when we generated the keystore.

javax.net.ssl.trustStore

Points to the truststore file we generated before.

javax.net.ssl.trustStorePassword

Provides the password we used when we generated the truststore.

gov.nist.javax.sip.TLS_CLIENT_PROTOCOLS

This is important because sipp supports only TLSv1 and so we need to restrict the protocols only to that.

gov.nist.javax.sip.TLS_CLIENT_AUTH_TYPE

If Enabled, used to request and require client certificate authentication: the connection will terminate if no suitable client certificate is presented. If Want, used to request client certificate authentication, but keep the connection if no authentication is provided. If Disabled or DisabledAll does not use authentication.

javax.net.debug

SSL will provide some extra debugging information for the SSL if uncomment it.

isRemoteServerSsl

Should smpp and http use secured connection towards servers or regular.

isTransportWs

Enables ws/wss transport in Restcomm Load Balancer.

statisticPort

Port for statistic

The remaining keys and properties in the configuration properties file can be used to tune the JAIN SIP stack, but are not specifically required for load balancing. To assist with tuning, a comprehensive list of implementing classes for the SIP Stack is available from the Interface SIP Stack page on nist.gov . For a comprehensive list of properties associated with the SIP Stack implementation, refer to Class SipStackImpl page on nist.gov .

If Restcomm Load Balancer is behind firewall you will need to open all the required ports. The default ports are: TCP: 2000, 2001, 2080, 5060, 5065, 8000 UDP: 5060, 5065

2. Configure logging

   The Restcomm Load Balancer uses Log4J as a logging mechanism. You can configure it through the typical log4j xml configuration file and specify the path as follows `-DlogConfigFile=./log4j.xml` . Please refer to Log4J documentation for more information on how to configure the logging. A shortcut exists if you want to switch between INFO/DEBUG/WARN logging levels. The JVM option `-DlogLevel=DEBUG` will allow you to switch all loggig categories to the specified log level.

Converged Load Balancing

Apache HTTP Load Balancer

The Restcomm Restcomm Load Balancer can work in concert with HTTP load balancers such as `mod_jk` . Whenever an HTTP session is bound to a particular node, an instruction is sent to the Restcomm Load Balancer to direct the SIP calls from the same application session to the same node.

It is sufficient to configure `mod_jk` to work for HTTP in JBoss in order to enable cooperative load balancing. Restcomm will read the configuration and will use it without any extra configuration. You can read more about configuring `mod_jk` with JBoss in your JBoss Application Server documentation.

Integrated HTTP Load Balancer

To use the integrated HTTP Load Balancer, no extra configuration is needed. If a unique `jvmRoute` is specified and enabled in each application server, it will behave exactly as the apache balancer. If `jvmRoute` is not present, it will use the session ID as a hash value and attempt to create a sticky session. The integrated balancer can be used together with the apache balancer at the same time.

In addition to the apache behavior, there is a consistent hash balancer algorithm that can be enabled for both HTTP and SIP messages. For both HTTP and SIP messages, there is a configurable affinity key, which is evaluated and hashed against each unassigned request. All requests with the same hash value will always be routed to the same application server node. For example, the SIP affinity key could be the callee user name and the HTTP affinity key could the “appsession” HTTP

GET parameter of the request. If the desired behaviour group these requests, we can just make sure the affinity values (user name and GET parameter) are the same.

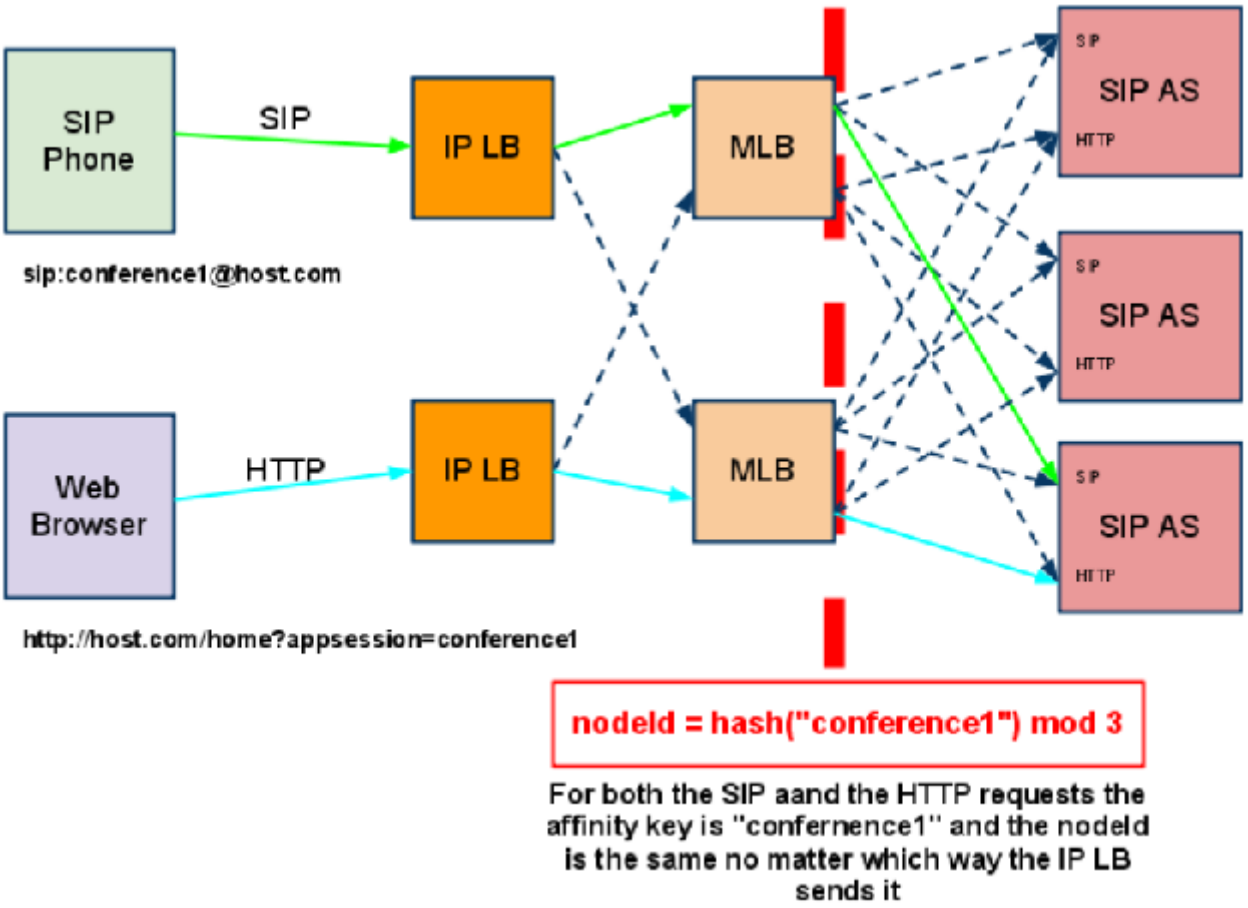

Figure 2. Ensuring SIP and HTTP requests are being grouped by commonaffinity value.

Running

Procedure: Running the SIP/SMPP Load Balancer and SIP Server Nodes

1. Start the SIP/SMPP Load Balancer

   Start the SIP/SMPP load balancer, ensuring the Configuration Properties file (*lb.properties* in this example) is specified. In the Linux terminal, or using the Windows Command Prompt, the SIP and SMPP Load Balancers are started by issuing a command similar to this one:

   ```
   java -DlogConfigFile=./lb-log4j.xml -jar sip-balancer-jar-with-dependencies.jar
   -mobicents-balancer-config=lb-configuration.properties
   ```

   Executing the Restcomm Load Balancer produces output similar to the following example:

```
home]$ java -DlogConfigFile=./lb-log4j.xml -jar sip-balancer-jar-with-
dependencies.jar -mobicents-balancer-config=lb-configuration.properties
2016-02-08 15:54:28,036 INFO main - nodeTimeout=8400
2016-02-08 15:54:28,038 INFO main - heartbeatInterval=150
2016-02-08 15:54:28,039 INFO main - Node registry starting...
2016-02-08 15:54:28,103 INFO main - RMI heartbeat listener bound to internalHost,
port 2000
2016-02-08 15:54:28,103 INFO main - Node expiration task created
2016-02-08 15:54:28,103 INFO main - Node registry started
2016-02-08 15:54:28,130 INFO main - value -1000 will be used for
reliableConnectionKeepAliveTimeout stack property
2016-02-08 15:54:28,131 INFO main - Setting Stack Thread priority to 10
2016-02-08 15:54:28,134 INFO main - using Disabled tls auth policy
2016-02-08 15:54:28,159 WARN main - using default tls security policy
2016-02-08 15:54:28,162 WARN main - Using default keystore type jks
2016-02-08 15:54:28,162 WARN main - Using default truststore type jks
2016-02-08 15:54:28,173 INFO main - the sip stack timer
gov.nist.javax.sip.stack.timers.DefaultSipTimer has been started
2016-02-08 15:54:28,325 INFO main - Sip Balancer started on external address
127.0.0.1, external port : 5060, internalPort : 5065
2016-02-08 15:54:28,365 INFO main - HTTP LB listening on port 2080
2016-02-08 15:54:28,377 INFO main - HTTPS LB listening on port 2081
2016-02-08 15:54:28,432 INFO main - SMPP Load Balancer started at 127.0.0.1 : 2776
2016-02-08 15:54:28,433 INFO main - SMPP Load Balancer uses port : 2876 for TLS
clients.
```

The output shows the IP address on which the Restcomm Load Balancer is listening, as well as the external and internal listener ports.

2. Configure SIP/SMPP Server Nodes

   The information about configuring your SIP Server, SMPP Server, SIP Servlets or JAIN SLEE, is in the respective server User Guide.

3. Start Load Balancer Client Nodes

   Start all SIP and SMPP load balancer client nodes.

Testing

To test load balancing, the same application must be deployed manually on each node, and two SIP Softphones must be installed.

Procedure: Testing Load Balancing with Sip Servlets

1. Deploy an Application

   Ensure that for each node, the DAR file location is specified in the *server.xml* file.

   Deploy the Location service manually on both nodes.

2. Start the "Sender" SIP softphone

   Start a SIP softphone client with the SIP address of `sip:sender@sip-servlets-com` , listening on port 5055. The outbound proxy must be specified as the sip-balancer (http://127.0.0.1:5060)

3. Start the "Receiver" SIP softphone

   Start a SIP softphone client with the SIP address of `sip:receiver-failover@sip-servlets-com` , listening on port 5090.

4. Initiate two calls from "Sender" SIP softphone

   Initiate one call from `sip:sender@sip-servlets-com` to `sip:receiver-failover@sip-servlets-com` . Tear down the call once completed.

   Initiate a second call using the same SIP address, and tear down the call once completed. Notice that the call is handled by the second node.

Procedure: Testing Load Balancing with JAIN SLEE and SIP RA

1. Deploy SIP RA
2. Configure the JAIN SIP HA properties for load balancing according to the JAIN SLEE User Guide
3. Deploy a sample application
4. Run the sample scenario for the application using the Restcomm Load Balancer

Stopping

Assuming that you started the JBoss Application Server as a foreground process in the Linux terminal, the easiest way to stop it is by pressing the key combination in the same terminal in which you started it.

This should produce similar output to the following:

```
^C2016-02-08 16:16:59,788 INFO Thread-142 - Stopping the sip forwarder
2016-02-08 16:16:59,789 INFO Thread-142 - Removing the following Listening Point
gov.nist.javax.sip.ListeningPointImpl@40185808
2016-02-08 16:16:59,791 INFO NioSelector-TLS-127.0.0.1/5061 - Selector is closed
2016-02-08 16:16:59,791 INFO Thread-142 - Removing the following Listening Point
gov.nist.javax.sip.ListeningPointImpl@a440543
2016-02-08 16:16:59,796 INFO Thread-142 - Removing the following Listening Point
gov.nist.javax.sip.ListeningPointImpl@a0d78e9
2016-02-08 16:16:59,796 INFO NioSelector-TCP-127.0.0.1/5060 - Selector is closed
2016-02-08 16:16:59,796 INFO Thread-142 - Removing the sip provider
2016-02-08 16:16:59,797 INFO Thread-142 - Removing the following Listening Point
gov.nist.javax.sip.ListeningPointImpl@1ce21add
2016-02-08 16:16:59,798 INFO NioSelector-TLS-127.0.0.1/5066 - Selector is closed
2016-02-08 16:16:59,798 INFO Thread-142 - Removing the following Listening Point
gov.nist.javax.sip.ListeningPointImpl@2a859dc9
2016-02-08 16:16:59,800 INFO Thread-142 - Removing the following Listening Point
gov.nist.javax.sip.ListeningPointImpl@63a80928
2016-02-08 16:16:59,804 INFO NioSelector-TCP-127.0.0.1/5065 - Selector is closed
2016-02-08 16:16:59,808 INFO Thread-142 - Removing the sip provider
2016-02-08 16:16:59,808 INFO Thread-142 - the sip stack timer
gov.nist.javax.sip.stack.timers.DefaultSipTimer has been stopped
2016-02-08 16:17:00,809 INFO Thread-142 - the sip stack timer
gov.nist.javax.sip.stack.timers.DefaultSipTimer has been stopped
2016-02-08 16:17:01,839 INFO Thread-142 - Sip forwarder SIP stack stopped
2016-02-08 16:17:01,839 INFO Thread-142 - Stopping the http forwarder
2016-02-08 16:17:01,953 INFO Thread-142 - Stopping the SMPP balancer
2016-02-08 16:17:01,957 INFO Thread-142 - SMPP Load Balancer stopped at 127.0.0.1
2016-02-08 16:17:01,958 INFO Thread-142 - Unregistering the node registry
2016-02-08 16:17:01,958 INFO Thread-142 - Unregistering the node adapter
2016-02-08 16:17:01,962 INFO Thread-142 - Stopping the node registry
2016-02-08 16:17:01,962 INFO Thread-142 - Stopping node registry...
2016-02-08 16:17:01,963 INFO Thread-142 - Node Expiration Task cancelled true
2016-02-08 16:17:01,963 INFO Thread-142 - Node registry stopped.
```

Uninstalling

To uninstall the SIP and SMPP load balancers, delete the JAR file you installed.

SIP Load Balancing Basics

All User Agents send SIP messages, such as `INVITE` and `MESSAGE`, to the same SIP URI (the IP address and port number of the Restcomm Load Balancer on the WAN). The Load Balancer then parses, alters, and forwards those messages to an available node in the cluster. If the message was sent as a part of an existing SIP session, it will be forwarded to the cluster node which processed that User Agent's original transaction request.

The SIP Server that receives the message acts upon it and sends a response back to the Restcomm Load Balancer. The Restcomm Load Balancer reparses, alters and forwards the message back to the

original User Agent. This entire proxying and provisioning process is carried out independent of the User Agent, which is only concerned with the SIP service or application it is using.

By using the Load Balancer, SIP traffic is balanced across a pool of available SIP Servers, increasing the overall throughput of the SIP service or application running on either individual nodes of the cluster. In the case of a Restcomm server with `</distributed>` capabilities, load balancing advantages are applied across the entire cluster.

The Restcomm Load Balancer is also able to failover requests mid-call from unavailable nodes to available ones, thus increasing the reliability of the SIP service or application. The Load Balancer increases throughput and reliability by dynamically provisioning SIP service requests and responses across responsive nodes in a cluster. This enables SIP applications to meet the real-time demand for SIP services.

HTTP Load Balancing Basics

In addition to the SIP load balancing, there are several options for coordinated or cooperative load balancing with other protocols such as HTTP.

Typically, a JBoss Application Server will use apache HTTP server with mod_jk, mod_proxy, mod_cluster or similar extension installed as an HTTP load balancer. This apache-based load balancer will parse incoming HTTP requests and will look for the session ID of those requests in order to ensure all requests from the same session arrive at the same application server.

By default, this is done by examining the `jsessionid` HTTP cookie or GET parameter and looking for the `jvmRoute` assigned to the session. The typical `jsessionid` value is of the form `<sessionId>.<jvmRoute>` . The very first request for each new HTTP session does not have a session ID assigned; the apache routes the request to a random application server node.

When the node responds it assigns a session ID and `jvmRoute` to the response of the request in a HTTP cookie. This response goes back to the client through apache, which keeps track of which node owns each `jvmRoute` . Once the very first request is served this way, the subsequent requests from this session will carry the assigned cookie, and the apache load balancer will always route the requests to the node, which advertised itself as the `jvmRoute` owner.

Instead of using apache, an integrated HTTP Load Balancer is also available. The Restcomm Load Balancer has a HTTP port where you can direct all incoming HTTP requests. The integrated HTTP load balancer behaves exactly like apache by default, but this behavior is extensible and can be overridden completely with the pluggable balancer algorithms. The integrated HTTP load balancer is much easier to configure and generally requires no effort, because it reuses most SIP settings and assumes reasonable default values.

Unlike the native apache, the integrated HTTP Load Balancer is written completely in Java, thus a performance penalty should be expected when using it. However, the integrated HTTP Balancer has an advantage when related SIP and HTTP requests must stick to the same node.

SMPP Load Balancing Basics

SMPP load balancer is used for balancing load from SMPP clients (ESME) to SMPP servers (SMSC) based on round-robin algorithm. Because of SMPP protocol based on an application layer TCP/IP connection between the ESME and SMSC and is initiated by the ESME, we should use connection handling. SMPP load balancer includes three main parts: server part, client part and dispatcher. When new clients connect to SMPP load balancer, application creates instances of ClientConnectionImpl (client part) and ServerConnectionImpl (server part) classes that bind by session ID. They are used to establish a connection between client (ESME) and server (SMSC). When connection established it can be used in both directions: ESME and SMSC can initiate new messages or send back messages through the SMPP load balancer. Dispatcher is used for logical connection of client and server parts of SMPP load balancer. Each packet that is received by the client part, the dispatcher throws to server part (unless error is detected) and vice versa.

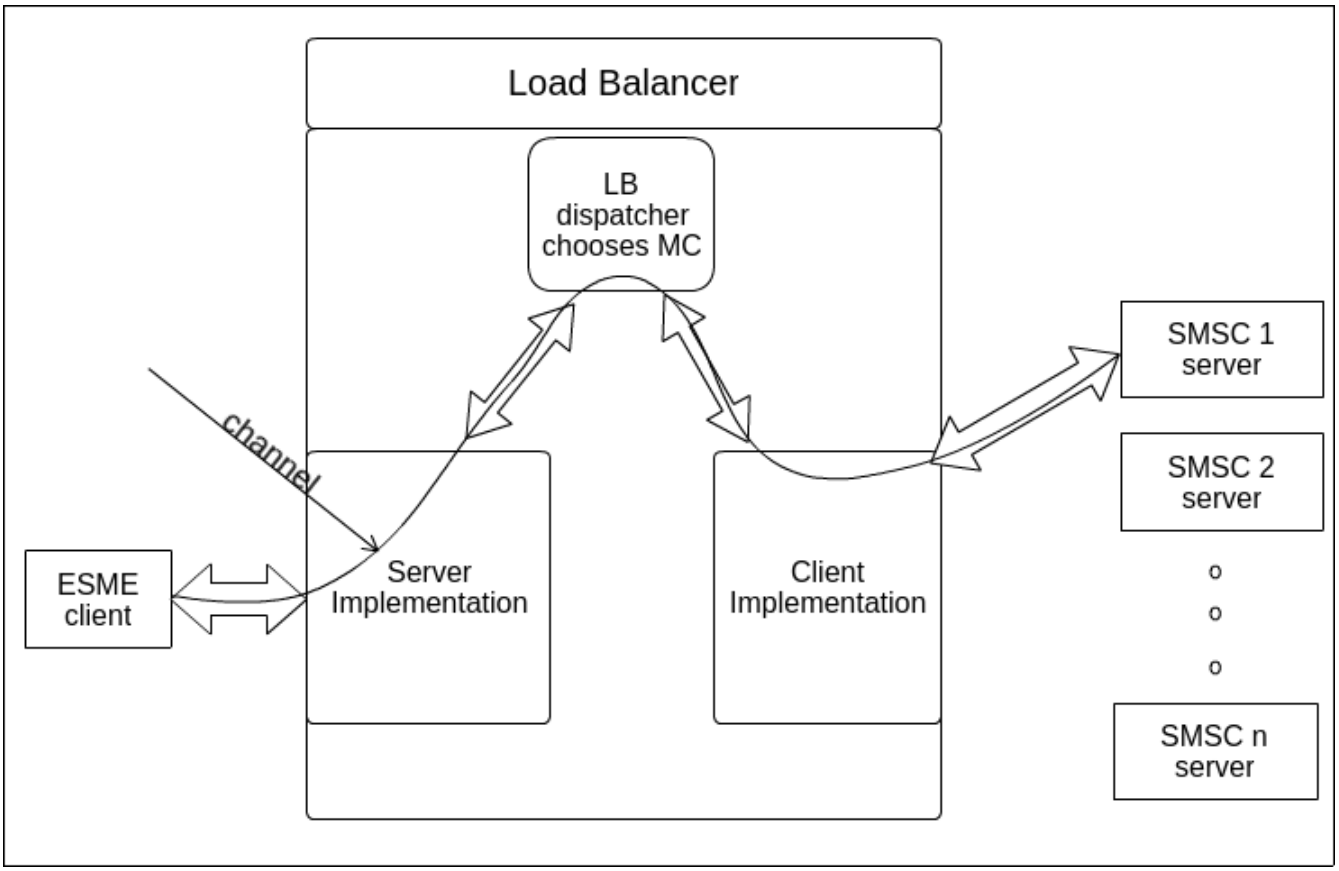

Figure 3. SMPP load balancer diagram

Main goal of application is to reduce the load on servers (SMSC) and simply transfer packets between client (ESME) and server(SMSC). But also it can inspect the received packets for correct command ID and will not send incorrect packets forward instead turn them back.

When connection to server drops, SMPP load balancer can reconnect (rebind) to the next working server. During this process (reconnect) it turns back all received packets until the new connection is established. If there are no established connections with the servers, the client connection is closed and vice versa.

Pluggable balancer algorithms

The SIP/HTTP Load Balancer exposes an interface to allow users to customize the routing decision making for special purposes. By default there are three built-in algorithms. Only one algorithm is active at any time and it is specified with the `algorithmClass` property in the configuration file.

It is up to the algorithm how and whether to support distributed architecture or how to store the information needed for session affinity. The algorithms will be called for every SIP and HTTP request and other significant events to make more informed decisions.

Users must be aware that by default requests explicitly addressed to a live server node passing through the load balancer will be forwarded directly to the server node. This allows for pre-specified routing use-cases, where the target node is known by the SIP client through other means. If the target node is dead, then the node selection algorithm is used to route the request to an available node.

The following is a list of the built-in algorithms:

org.mobicents.tools.sip.balancer.CallIDAffinityBalancerAlgorithm

This algorithm is not distributable. It selects nodes randomly to serve a give Call-ID extracted from the requests and responses. It keeps a map with `Call-ID → nodeId` associations and this map is not shared with other load balancers which will cause them to make different decisions. For HTTP it behaves like apache.

org.mobicents.tools.sip.balancer.HeaderConsistentHashBalancerAlgorithm

This algorithm is distributable and can be used in distributed load balancer configurations. It extracts the hash value of specific headers from SIP and HTTP messages to decide which application server node will handle the request. Information about the options in this algorithms is available in the balancer configuration file comments.

org.mobicents.tools.sip.balancer.PersistentConsistentHashBalancerAlgorithm

This algorithm is distributable and is similar to the previous algorithm, but it attempts to keep session affinity even when the cluster nodes are removed or added, which would normally cause hash values to point to different nodes.

org.mobicents.tools.sip.balancer.ClusterSubdomainAffinityAlgorithm

This algorithm is not distributable, but supports grouping server nodes to act as a subcluster. Any call of a node that belongs to a cluster group will be preferentially failed over to a node from the same group. To configure a group you can just add the `subclusterMap` property in the load balancer properties and listing the IP addresses of the nodes. The groups are enclosed in parentheses and the IP addresses are separate by commas as follows:

```
subclusterMap=( 192.168.1.1, 192.168.1.2 ) ( 10.10.10.10,20.20.20.20, 30.30.30.30)
```

The nodes specified in a group do not have to alive and nodes that are not specified are still allowed to join the cluster. Otherwise the algorthim behaves exactly as the default Call-ID affinity algorthim.

Distributed load balancing

When the capacity of a single load balancer is exceeded, multiple load balancers can be used. With the help of an IP load balancer the traffic can be distributed between all SIP/HTTP load balancers based on some IP rules or round-robin. With consistent hash and `jvmRoute` -based balancer algorithms it doesn't matter which SIP/HTTP load balancer will process the request, because they would all make the same decisions based on information in the requests (headers, parameters or cookies) and the list of available nodes. With consistent hash algorithms there is no state to be preserved in the SIP/HTTP balancers.

[bidirectional distributed sip lb] | *images/bidirectional-distributed-sip-lb.gif*

Figure 4. Example deployment: IP load balancers serving both directions forincoming/outgoing requests in a cluster

Implementation of the Restcomm Load Balancer

Each individual Restcomm SIP Server in the cluster is responsible for contacting the Restcomm Load Balancer and relaying its health status and regular "heartbeats".

From these health status reports and heartbeats, the Restcomm Load Balancer creates and maintains a list of all available and healthy nodes in the cluster. The Load Balancer forwards SIP requests between these cluster nodes, providing that the provisioning algorithm reports that each node is healthy and is still sending heartbeats.

If an abnormality is detected, the Restcomm Load Balancer removes the unhealthy or unresponsive node from the list of available nodes. In addition, mid-session and mid-call messages are failed over to a healthy node.

The Restcomm Load Balancer first receives SIP requests from endpoints on a port that is specified in its Configuration Properties configuration file. The Restcomm Load Balancer, using a round-robin algorithm, then selects a node to which it forwards the SIP requests. The Load Balancer forwards all same-session requests to the first node selected to initiate the session, providing that the node is healthy and available.

Implementation of the SMPP Load Balancer

SMPP load balancer implements timers: enquire link timer, session initialization timer and response timer for connection handling. Timers of SMPP load balancer have next behaviour:

- Session initialization timer disconnects client (ESME) if it does not send bind request in defined time;
- Response timer sends response with system error if sender does not receive response in defined time;
- Enquire link timer with a fixed rate checks the connections with client (ESME) and server

(SMSC).

Server part of SMPP load balancer has next states:

- OPEN - it can receive only bind requests from client (ESME);
- BINDING - it can't receive any messages, in this state we wait for client's response;
- BOUND - it can receive all PDU packets from client (ESME), which he can send according SMPP protocol, except bind requests;
- REBINDING - it can also receive all PDU packets from client (ESME), but returns them back, because the client part at this time is trying to reconnect to server;
- UNBINDING - it can receive only unbind response from client (ESME);
- CLOSED - it can't receive any messages, this is last state of life cycle, which indicate that connection is closed.

Client part of SMPP load balancer has next states:

- INITIAL - it can't receive any messages, this is first state of life cycle, at this state the client part is trying to connect to the server (SMSC) and if the connection is successful state changes to OPEN;
- OPEN - it can't receive any messages, at this state the client part sends a bind request to the server (SMSC), and changes state to binding;
- BINDING - it can receive only bind response from server, and if response does not have errors, the client part changes ones state to bound;
- BOUND - it can receive all packets from server (SMSC), which can be sent according SMPP protocol, except unbind response;
- REBINDING - if connection drops to the server (SMSC), the client part changes ones state to rebinding until reconnect. If reconnect fails, connection is closed;
- UNBINDING - it can receive unbind response from server only, after which state changes to closed state;
- CLOSED - it can't receive any messages, this is the last state of the life cycle, which indicates that the connection is closed.

SIP Message Flow

The Restcomm Load Balancer appends itself to the `Via` header of each request, so that returned responses are sent to the SIP Balancer before they are sent to the originating endpoint.

The Load Balancer also adds itself to the path of subsequent requests by adding Record-Route headers. It can subsequently handle mid-call failover by forwarding requests to a different node in the cluster if the node that originally handled the request fails or becomes unavailable. The Restcomm Load Balancer immediately fails over if it receives an unhealthy status, or irregular heartbeats from a node.

Additionally, Restcomm Load Balancer will add two custom header containing the initial remote

address and port of the sip client for every REGISTER request or requests with content.

- X-Sip-Balancer-InitialRemoteAddr
- X-Sip-Balancer-InitialRemotePort

Application can use these two headers to have the correct location of the sip client that sent the REGISTER request.

In advanced configurations, it is possible to run more than one Restcomm Load Balancer. Simply edit the balancers connection string in your SIP Server - the list is separated with semi-colon.

Basic IP and Port Cluster Configuration describes a basic IP and Port Cluster Configuration. In the diagram, the Restcomm Load Balancer is the server with the IP address of `192.168.1.1` .

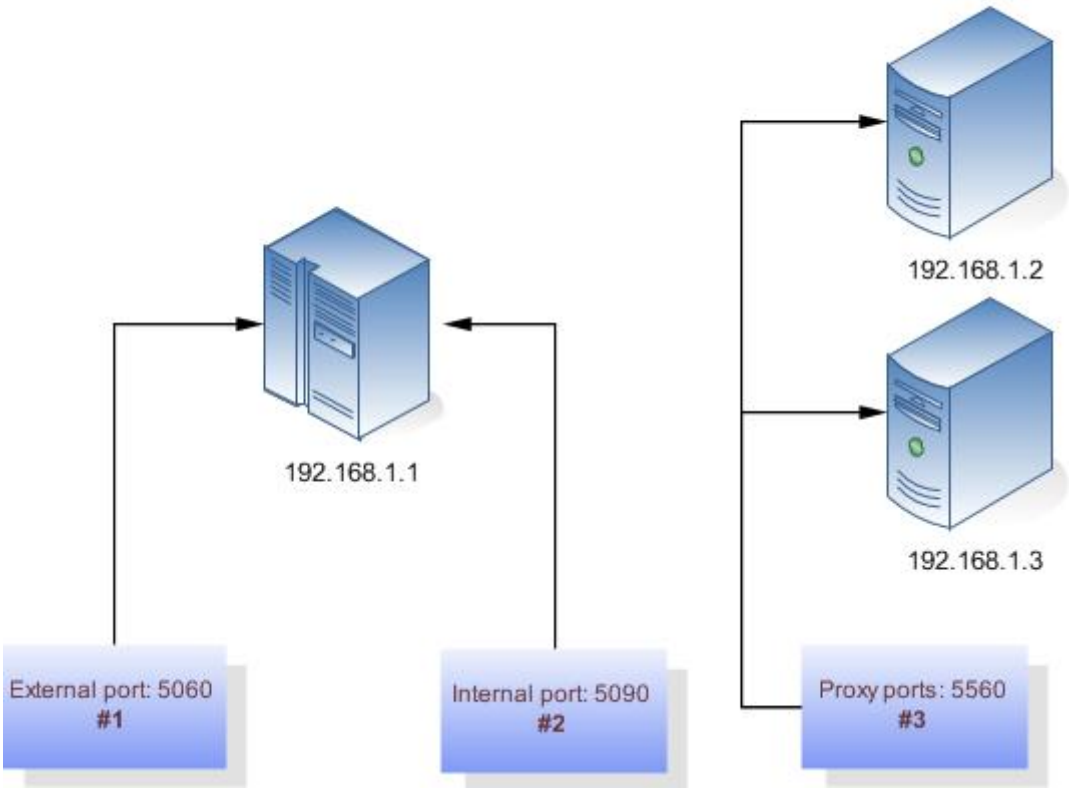

Figure 5. Basic IP and Port Cluster Configuration

Using the Load Balancer with Third-Party SIP Servers

The load balancer only forwards requests to servers that send heartbeat signals. A third party server can send metadata using a SIP OPTIONS or SIP INFO message towards the internal port of the Restcomm Load Balancer. For security reasons heartbeat messages arriving at the external entry-point will be ignored and using a single internal and external entry-point is not allowed. The third party SIP server must advertise it's metadata in the SIP message contents.

For example, this request will advertise a SIP server listening on 127.0.0.1:5070, both TCP and UDP .

```
OPTIONS sip:X@127.0.0.1:5065;lr SIP/2.0
Call-ID: 2298704dc4d6706e53cb61123ea7833e@127.0.0.1
CSeq: 1 OPTIONS
From: <sip:third-party-app-server-heartbeating-service@sip-servlets.com>;tag=4481411
To: <sip:sip-load-balancer@sip-servlets.com>
Via: SIP/2.0/UDP 127.0.0.1:5070;branch=z9hG4bK-373335-ec2c7452cfd0130bd409ba4f8ea5f54e
Max-Forwards: 70
Contact: <sip:sender@127.0.0.1:4060;transport=udp;lr>
Route:
<sip:lbaddress_InternalPort@127.0.0.1:5065;node_host=127.0.0.1;node_port=5070;lr>
Content-Type: text/plain;charset=UTF-8
Mobicents-Heartbeat: 1
Content-Length: 54

tcpPort=5070
udpPort=5070
hostname=sipHeartbeat
ip=127.0.0.1
```

The important headers to be filled in this request are `Mobicents-Heartbeat` , the Route header with `;node_host=127.0.0.1;node_port=5070` and the message contents. The message contents are interpreted as properties of the `SIPNode` object representing the node in the load balancer and can be further interpreted by load balancing algorithms for load balancing purposes. The value of the `Mobicents-Heartbeat` header is arbitrary and reserved for future use, the presence of the header is sufficient to instruct the load balancer how to process the request.

All requests initiated by the SIP Server must have the following hint in their Route header `;node_host=127.0.0.1;node_port=5070` . This hint instructs the Restcomm Load Balancer that the dialog initated by the application server must stay on the node advertised in the hint. This function is cruicial when the direction of the requests withing the dialog is reversed.

Since this SIP request represents a heartbeat signal, it must be sent regularly at least once every 5 seconds (by default). Sending this request is responsibility of the third party server. The load balancer will respond to every heartbeat request with 200 OK immediately. The third party server must expect the OK response. If no response if received within a threshhold time then the third party SIP server must assume the Restcomm Load Balancer is not available and use another (backup) load balancer.

The regular Restcomm Load Balancer setting are still in effect for third party servers and you can expect the same behavior as the Mobicents-specific RMI heartbeat configuration.

If you are desgning a pluggable algorithm using the SIP metadata, you can access the properties passed in the SIP message contents using `SIPNode.getProperties()`

Enabling TLS/WSS for Load Balancers

Enabling TLS/WSS for Restcomm Load Balancer

For enabling TLS support for Restcomm Load Balancer, you should correctly specify next property:

- javax.net.ssl.keyStore;
- javax.net.ssl.keyStorePassword;
- javax.net.ssl.trustStore;
- javax.net.ssl.trustStorePassword;
- gov.nist.javax.sip.TLS_CLIENT_AUTH_TYPE
- gov.nist.javax.sip.TLS_CLIENT_PROTOCOLS;
- externalSecurePort;
- internalSecurePort;
- externalSecureIpLoadBalancerPort (if you are using IP load balancer);
- internalSecureIpLoadBalancerPort (if you are using IP load balancer)
- isTransportWs(enables WSS/WS instead of TLS/TCP).

Enabling HTTPS

For enabling HTTPS and Secure WebSockets support for HTTP load balancer, you should correctly specify next property (httpsPort property enables HTTPS support):

- httpsPort;
- javax.net.ssl.keyStore;
- javax.net.ssl.keyStorePassword;
- javax.net.ssl.trustStore;
- javax.net.ssl.trustStorePassword;
- isRemoteServerSsl (if remote HTTP server uses TLS).

Enabling TLS for SMPP load balancer

For enabling TLS support for SMPP load balancer, you should correctly specify next property (smppSslPort property enables TLS suport):

- smppSslPort;
- javax.net.ssl.keyStore;
- javax.net.ssl.keyStorePassword;
- javax.net.ssl.trustStore;
- javax.net.ssl.trustStorePassword;
- isRemoteServerSsl (if remote SMPP server uses TLS).

Getting statistic

Load balancers provide some statistic by Java Management Extensions and JSON. You should use url http://host:statisticPort/lbstat for getting JSON statistic data from load balancer. You can get next data:

1. SIP balancer statistic:
    a. Number of processed requests;
    b. Number of processed responses;
    c. Number of transferred bytes;
    d. Number of requests processed by method;
    e. Number of responses processed by status code;
    f. Number of active connections(if transport is UDP we can't get this value);
2. HTTP balancer statistic:
    a. Number of HTTP request;
    b. Number of bytes transferred to server;
    c. Number of bytes transferred to client;
    d. Number of requests processed by HTTP method;
    e. Number of responses processed by HTTP code;
    f. Number of active connections;
3. SMPP balancer statistic:
    a. Number of requests to server;
    b. Number of requests to client;
    c. Number of bytes transferred to server;
    d. Number of bytes transferred to client;
    e. Number of requests processed by SMPP Command ID;
    f. Number of responses processed by SMPP Command ID;
    g. Number of active connections;
4. Other data:
    a. Recent CPU usage for the Java Virtual Machine process;
    b. Amount of used memory of the heap in bytes.