Best Practices

Table of Contents

This chapter discusses Best Practices related to Restcomm SIP Servlets usage in real world deployments.

Restcomm SIP Servlets Performance Tips

Because the default profile of Restcomm SIP Servlets is targeted at a development environment, some tuning is required to make the server performance suitable for a production environment.

A useful presentation from OKI Japan

Tuning JBoss

To ensure the server is finely tuned for a production envirionment, certain configuration must be changed. Visit the JBoss Application Server Tuning wiki page to learn about optimization techniques.

While it is preferable to have a fast Application Server, most of the information doesn't apply to Restcomm . In summary, the most important optimization technique is to remove logs, leaving only what is required.

Check the log configuration file in the following location and review the information.

SIP Servlets Server Logging

Tuning Restcomm SIP Servlets

- *Congestion Control*: It is recommended that this feature is enabled to avoid overload of the server and that the *sipMessageQueueSize* and *memoryThreshold* parameters are tuned according to Concurrency and Congestion Control.
- *Concurrency* : *Default Value: None.* For better performance, it is recommended to leave this value set to `None`.

Tuning The JAIN SIP Stack

The stack can be fine-tuned by altering the SIP stack properties, defined in the external properties file specified by the *sipStackPropertiesFile* attribute as described in Configuring SIP Connectors and Bindings.

- *gov.nist.javax.sip.THREAD_POOL_SIZE*

  Default value: 64

  This thread pool is responsible for parsing SIP messages received from socket messages into objects.

  A smaller value will make the stack less responsive, since new messages have to wait in a queue for free threads. In UDP, this can lead to more retransmissions.

Large thread pool sizes result in allocating resources that are otherwise not required.

- *gov.nist.javax.sip.REENTRANT_LISTENER*

  Default value: true

  This flag indicates whether the SIP stack listener is executed by a single thread, or concurrently by the threads that parse the messages.

  Restcomm SIP Servlets expects this flag to be set to `true`, therefore do not change the value.

- *gov.nist.javax.sip.LOG_MESSAGE_CONTENT*

  Default value: true

  Set the parameter to `false` to disable message logging.

- *gov.nist.javax.sip.TRACE_LEVEL=0*

  Default value: 32.

  Set the parameter to `0` to disable JAIN SIP stack logging.

- *gov.nist.javax.sip.RECEIVE_UDP_BUFFER_SIZE=65536* *and* *gov.nist.javax.sip.SEND_UDP_BUFFER_SIZE=65536*

  Default value: 65536.

  Those properties control the size of the UDP buffer used for SIP messages. Under load, if the buffer capacity is overflown the messages are dropped causing retransmissions, further increasing the load and causing even more retransmissions.

- *gov.nist.javax.sip.MAX_MESSAGE_SIZE=10000*

  Default value: 10000.

  This property controls the maximum size of content that can be read for a SIP Message on UDP. The default is 65536. The average UDP message size is quite lower than this so reducing this property will benefit memory usage since a byte buffer of this size is created for every message received.

  It also defines the maximum size of content that a TCP connection can read. Must be at least 4K. Default is "infinity" — ie. no limit. This is to prevent DOS attacks launched by writing to a TCP connection until the server chokes.

- *gov.nist.javax.sip.TCP_POST_PARSING_THREAD_POOL_SIZE=30*

  Default value: 30.

  Use 0 or do not set this option to disable it. When using TCP, your phones/clients usually connect independently, creating their own TCP sockets. Sometimes however SIP devices are allowed to tunnel multiple calls over a single socket. This can also be simulated with SIPP by running "sipp

-t t1".

In the stack, each TCP socket has it's own thread. When all calls are using the same socket they all use a single thread, which leads to severe performance penalty, especially on multi-core machines. This option instructs the SIP stack to use a thread pool and split the CPU load between many threads. The number of the threads is specified in this parameter.

The processing is split immediately after the parsing of the message. It cannot be split before the parsing because in TCP the SIP message size is in the Content-Length header of the message and the access to the TCP network stream has to be synchronized.

Additionally, in TCP the message size can be larger. This causes most of the parsing for all calls to occur in a single thread, which may have impact on the performance in trivial applications using a single socket for all calls. In most applications it doesn't have performance impact. If the phones/clients use separate TCP sockets for each call, this option doesn't have much impact, except the slightly increased memory footprint caused by the thread pool. It is recommended to disable this option in this case by setting it 0 or not setting it at all. You can simulate multi-socket mode with "sipp -t t0". With this option also we avoid closing the TCP socket when something fails, because we must keep processing other messages for other calls. Note: This option relies on accurate Content-Length headers in the SIP messages. It cannot recover once a malformed message is processed, because the stream iterator will not be aligned any more. Eventually the connection will be closed.

The full list of JAIN SIP stack properties is available from the SIP Stack Properties Home Page and the full list of implementation specific properties are available from the SIP Stack Implementation Home Page.

Tuning The JVM

The following tuning information applies to Sun JDK 1.6, however the information should also apply to Sun JDK 1.5.

For more information on tuning Restcomm SIP Servlets performance, refer to the OKI Japan Presentation.

For more information on performance, refer to the Performance White Paper.

To pass arguments to the JVM, change *$JBOSS_HOME/bin/standalone.conf* (Linux) or *$JBOSS_HOME/bin/standalone.bat* (Windows).

- *Garbage Collection*

  JVM ergonomics automatically attempt to select the best garbage collector. The default behaviour is to select the throughput collector, however a disadvantage of the throughput collector is that it can have long pauses times, which ultimately blocks JVM processing.

  For low-load implementations, consider using the incremental, low-pause, garbage collector (activated by specifying `*-XX:+UseConcMarkSweepGC -XX:+CMSIncrementalMode*`). Many SIP applications can benefit from this garbage collector type because it reduces the retransmission

amount.

For more information please read: Garbage Collector Tuning

- *Heap Size*

  Heap size is an important consideration for garbage collection. Having an unnecessarily large heap can stop the JVM for seconds, to perform garbage collection.

  Small heap sizes are not recommended either, because they put unnecessary pressure on the garbage collection system.

Tuning The Operating System

The following tuning information is provided for Red Hat Enterprise Linux (RHEL) servers that are running high-load configurations. The tuning information should also apply to other Linux distributions.

After you have configured RHEL with the tuning information, you must restart the operating system. You should see improvements in I/O response times. With SIP, the performance improvement can be as high as 20%.

- *Large Memory Pages*

  Setting large memory pages can reduce CPU utilization by up to 5%.

  Ensure that the option `*-XX:+UseLargePages*` is passed and ensure that the following Java HotSpot™ Server VM warning does not occur:

  `Failed to reserve shared memory (errno = 22)" when starting JBoss.` It means that the number of pages at OS level is still not enough.

  To learn more about large memory pages, and how to configure them, refer to Java Support for Large Memory Pages and Andrig's Miller blog post.

- *Network buffers*

  You can increase the network buffers size by adding the following lines to your */etc/sysctl.conf* file:

- `net.core.rmem_max = 16777216`
- `net.core.wmem_max = 16777216`
- `net.ipv4.tcp_rmem = 4096 87380 16777216`
- `net.ipv4.tcp_wmem = 4096 65536 16777216`
- `net.core.netdev_max_backlog = 300000`
- Execute the following command to set the network interface address:

```
sudo ifconfig [eth0] txqueuelen 1000 #
```

Replace [eth0] with the correct name of the actual network interface you are setting up.

NAT Traversal

In a production environment, it is common to see SIP and Media data passing through different kinds of Network Address Translation (NAT) to reach the required endpoints. Because NAT Traversal is a complex topic, refer to the following information to help determine the most effective method to handle NAT issues.

STUN

STUN (Session Traversal Utilities for NAT) is not generally considered a viable solution for enterprises because STUN cannot be used with symmetric NATs.

Most enterprise-grade firewalls are symmetric, therefore STUN support must be provided in the SIP Clients themselves.

Most of the proxy and media gateways installed by VoIP providers recognize the public IP address the packets have originated from. When both SIP end points are behind a NAT, they can act as gateways to clients behind NAT.

TURN

TURN (Traversal Using Relay NAT) is an IETF standard, which implements media relays for SIP end-points. The standard overcomes the problems of clients behind symmetric NATs which cannot rely on STUN to solve NAT traversal.

TURN connects clients behind a NAT to a single peer, providing the same protection offered by symmetric NATs and firewalls. The TURN server acts as a relay; any data received is forwarded.

This type of implementation is not ideal. It assumes the clients have a trust relationship with a TURN server, and a request session allocation based on shared credentials.

This can result in scalability issues, and requires changes in the SIP clients. It is also impossible to determine when a direct, or TURN, connection is appropriate.

ICE

ICE (Interactive Connection Establishment) leverages both STUN and TURN to solve the NAT traversal issues.

It allows devices to probe for multiple paths of communication, by attempting to use different port numbers and STUN techniques. If ICE support is present in both devices, it is quite possible that the devices can initiate and maintain communication end-to-end, without any intermediary media relay.

Additionally, ICE can detect cases where direct communication is impossible and automatically

initiate fall-back to a media relay.

ICE is not currently in widespread use in SIP devices, because ICE capability must be embedded within the SIP devices.

Depending on the negotiated connection, a reINVITE may be required during a session, which adds more load to the SIP network and more latency to the call.

If the initiating ICE client attempts to call a non-ICE client, then the call setup-process will revert to a conventional SIP call requiring NAT traversal to be solved by other means.

Other Approaches

While the above is a good solution to circumvent NAT issues. There might be cases where it is not possible to use those solutions at all.

Other approaches include using proxy and media that can act as gateways, Session Border Controllers, enhanced Firewall with Application Layer Gateway (ALG) and Tunnelling.

Here is more information on Session Border Controllers and how they can resolve NAT issues when above solutions are not possible