Test procedure - MAS

Content

History

Version	Date	Adjustments
A	2007-05-25	Updated after review, first version.
B	2007-11-05	Updated with AMR

C	2007-12-07	Minor changes. Added capacity chapter in "TIPS & TRICKS".

1 Introduction

This document describes some of the test procedures and tools used for testing the MAS component. The document is primarily intended to support component test activities, but the information might be found useful in any test phase.

2 Deployment and tools

2.1 Deployment and Environment

The deployment of the test object and the surrounding components/simulated components necessary to perform a particular type of test differs depending on the type of test that is about to be performed, lab equipment availability and the possibility of stubbing actual system components interacting with the MAS.

2.2 Traffic generation

The MAS requires external stimuli to perform its traffical tasks. The stimuli currently include SIP and RTP traffic exchanged with a telephony client (e.g. a SIP softphone or a PSTN gateway) and service requests sent by system internal entities. The latter includes e.g. outdial notification service from the NTF component over the XMP.

2.2.1 SIP

SIP traffic is mainly generated as single calls using a softphone, a bulk SIP traffic generation tool or a PSTN gateway (MTG) which employs SIP on its IP interface.

2.2.1.1 Manual single calls using softphone

For voice/video tests that can be performed manually, a supported SIP softphone (e.g. "Kapanga" [1]) can be used to generate a call. Non-standard MAS SIP parameters such as test=on must then be appended to the SIP URI (";" separator), with a MAS at 10.11.0.32 use e.g.:

```
10.11.0.32;test=on;calling=4321;calling_privacy=off;called=1234;redir
=5678;redir_privacy=on;redir_cause=user-busy
```

Using test parameters require special settings in the MAS configuration for the call manager; refer to the MAS O&M guide for details. Table 1 (from the New Development tips and tricks page) summarizes how to use these parameters:

Table 1: Test parameter usage

Parameter	Possible values	Description

test=	on, off	Activate reading test parameters. Make sure the feature is enabled in mas.xml (default=false).
calling=	123456789	A-nr/ANI
calling_privacy=	on, off	If the parameter is set to *on* the presentation is restricted. If the parameter is set to *off* the presentation is allowed. Otherwise, the presentation indicator will be considered to be unknown by the MAS.
called=	123456789	B-nr/DNIS. Should typically match the deposit or retrieval DNIS configured in vva.xml
redir=	123456789	C-nr / RDNIS
redir_privacy=	on, off	If the parameter is set to *on* the presentation is restricted. If the parameter is set to *off,* the presentation is allowed. Otherwise, the presentation indicator will be considered to be unknown by the MAS.
redir_cause=	user-busy, away, deflection, do-not-disturb, follow-me, no-answer, out-of-service, time-of-day, unavailable, unconditional, unknown	The redirecting cause.

If MAS is using AMR as codec you must configure Kapanga to use AMR by entering Configuration->CallControlSettings and adding AMR in the offered codec list instead of G711. The correct bitrate and packet size must also be set to 20ms and 12200 bps by entering Configuration->CodecSettings and selecting >AMR. To be able to run AMR a license must be installed on Kapanga.
A more thorough description of these parameters and the SIP interface can be found e.g. in the system level SIP IWD or the MAS call manager design documentation.

2.2.1.2 Automated bulk softphone calls

Automated scripted voice/video calls (single or multiple channels) can be generated in software using the "sipp" tool [2], which is available on "gulltopp" with some default scripts. This tool is controlled by xml-style scenario files. The scenario files are also typically augmented by .csv files, containing e.g. telephone number ranges (typically ANI and RDNIS). Furthermore, the scenarios are supported by .pcap files, containing DTMF/voice/video RTP stream fragments, which may be sent to the MAS. Many scripts are "test=on"-based, but scripts which emulate the SIP signaling of specific gateways (e.g. Cisco) or which make use of IMS-specific headers can also be implemented.

A simple example of using the sipp tool:

 # export LD_LIBRARY_PATH=./

 # ./sipp –sf voice_deposit.xml –inf telNumList.csv –r 1 –rp 250 –l 20 –trace_err mas_host_name

This example would start the voice_deposit.xml scenario towards mas_host_name, with the scenario using numbers from telNumList.csv. One channel is started every 250 ms, until the limit of 20 simultaneous channels has been reached, and errors will be logged.

2.2.1.3 Bulk Hammer + MTG calls

Automated scripted SS7 voice calls can also be defined in Empirix hammer scripts if a full Hammer/MTG setup is available. In this case, the test traffic originates/terminates outside the MoIP system, simulating PSTN access. The MTG will bridge between the PSTN SS7 and the SIP/IP-telephony domains, as is normally the case in current system installations. The details on MTG, hammer unit and hammer script setup/preparation is out of scope for this document.

On a freshly rebooted Hammer unit, the basic startup procedure is to start the "Hammer Configurator" application, enter "HW Configurator", press "load boards", enter "system status" and press "start". The link status should now be "in service" (check "SS7 management"). Then start the "Hammer test builder" application to start the scripts (with the appropriate modifications). The scripts (e.g. deposit, retrieve, slamdown and outdial answer scripts) are usually found under "scripts" in the left panel. The scripts are typically started per-channel by a keep-alive script for each scenario. The keep-alive script for each scenario starts the correct scripts on the desired channels and keeps them running.

Measured response times experienced by the hammer scripts can be saved by enabling logging on a subset of the channels (include at least two deposit, two retrieve and one slamdown channel). The output log files for each channel can then be processed to extract the response time statistics for each call type using the appropriate post-processing scripts.

2.3 MAS/JVM process supervision

When the MAS is handling traffic under load, it is important to monitor and try to assess the status and health of the MAS JVM process. Since the MAS implementation is mainly in the Java domain, the majority of the code is executed by the 3PP JVM and is subject to the benefits/drawbacks of Java. The part of the MAS that performs media streaming (stream) is, however, partially implemented in C++. It is therefore loaded and executed as a native library in the JVM without the benefits/drawbacks of Java. The Java/native stream implementation parts interact using the Java Native Interface (JNI).

2.3.1 General

To log e.g. CPU and memory usage for the entire MAS JVM process while running, use e.g.:

 # prstat –p <JVM_PID> 60 > prstat.log&

Use cut/grep/… to extract the columns of interest, import into excel.

Various statistics can be sampled over time per lightweight process (LWP) in the running MAS Java Virtual Machine (JVM). The Solaris 10 Operating Environment™ maintains a one-to-one thread to LWP mapping. The threads in the MAS JVM include java threads (which execute the Java code in the MAS) as well as native threads (which execute purely native code in the MAS), e.g. the native part of Stream and e.g. the GC (Garbage Collector) threads of the JVM itself.

Many important threads will live for the lifetime of the JVM, and will therefore keep their LWP mapping, which is a useful fact when trying to track their activities.

Create a pstack for the JVM pid to help identify which thread is mapped to a particular LWP, c++filt (if available) can help clarify the content of the pstack file.

 # pstack <JVM_PID> > pstack_<JVM_PID>
 # pstack <JVM_PID> | c++filt > pstack_<JVM_PID>

If available, a mixed jstack (available in jdk, i.e. jstack –m <PID>) can provide a more easily interpreted output.

 # <JDK_HOME>/bin/jstack –m <JVM_PID> > jstack_<JVM_PID>

Sample micro-statistics, e.g. every 30th second (include up to 150 LWPs):

 # prstat –p <JVM_PID> -n 150 30 > prstat_mL.log&

After the sampling period, kill the prstat commands running in the background:

 # pkill prstat

Open the jstack/pstack file in an editor to identify the LWP IDs of threads of interest. The core JVM threads (e.g. GC threads) are often among the first 10 LWPs in the process. A set of stream threads should also be allocated per (virtual) CPU (each set is normally in 5 adjacent LWP IDs on a single CPU host, 1 for outbound and 4 for inbound RTP streams).

Sampled microstate information for each corresponding LWP in the MAS JVM from the sampled prstat_mL.log file using can be extracted, e.g. for LWP 2:

 # grep ”java/2$” prstat_mL.log > lwp_2.log

The file lwp_2.log can now be refined further to extract the desired data colums, e.g. for plotting over time (with the time scale ~30 seconds between samples).

To find out which LWP ID the most active threads in the process have, grep for a range of LWP IDs in prstat_mL.log instead of identifying the LWPs of interest through the jstack/pstack.

A range of other Solaris standard tools (such as DTrace, truss, plockstat, iostat, gcore, pmap, mdb etc) can be used for process-level diagnostics online as well as

offline, refer to e.g. [3] for a general description (aimed at T-1 multi-core CPUs in particular). DTrace scripts which have some general use (e.g. timing native or Java method calls, finding rouge method calls etc while under load in a more or less intrusive manner) have been developed. These can be tailored for specific investigations by the appropriate designer.

Basic guidelines:

- Log prstat statistics periodically to a file
  - Include a separate prstat for microstate information if desired
- Create a pstack, save to a file
  - Optionally demangle using c++filt
- Apply load
- When traffic has been running for a while, stop the prstat commands, extract process/thread data using the information in the pstack and using standard Unix commands
- Import the refined data into excel. Analyze further / plot.

2.3.2 Native memory

Major native memory leaks are easiest to spot by logging the process size of the MAS JVM and/or by performing "pmap <JVM_PID> > pmapX.log" periodically while running traffic, and observing the growth of specific memory areas (e.g. on the native heap, which should generally remain pretty stable).

If native memory problems (e.g. leaks) are suspected and need to be diagnosed further, the memory allocator (umem) can be used in debug mode (this will have a performance impact) by setting the appropriate options in the mas start script, add (in the beginning of the script) e.g.:

export LD_PRELOAD_32="libumem.so.1 libccrtpadapter.so"

and possibly, for a basic memory leak search example e.g.
export UMEM_DEBUG="audit=19"

A range of other umem options may be of interest, in particular when looking for other types of memory issues (e.g. corruption). E.g.:
export UMEM_OPTIONS="backend=mmap"
export UMEM_DEBUG="default,audit=45"
export UMEM_DEBUG="firewall=1"
export UMEM_LOGGING=transaction
export UMEM_LOGGING=audit
…

When the appropriate umem environment variables have been set, start traffic towards the MAS (which may now be very slow in its execution, depending on the umem flags used). The gcore command should then be used periodically on the MAS JVM PID while running traffic, to allow later tracking of the memory allocation development over time.

The core files can then be forwarded to an appropriate designer, who can analyze them further e.g. using the Solaris mdb command. The appropriate designer might use e.g. "::findleaks", "::umem_status", "::umem_verify", "::walk_thread", "::walk_bufctl", "::bufctl_audit", "::umem_status", "::umausers" and so on to investigate the core file further.

The usefulness of a core file depends on the options set before starting the MAS, so when debugging, the non-standard umem options may required to provide the necessary input. If non-standard libraries/jar-files ("black builds") are in used in tests reproducing faults, make sure to include these with any resulting core file to help with the analysis.

2.3.3 Core file handling

The MAS JVM occasionally cores without assistance. The resulting core files can normally be found in /apps/mas, sometimes there will also be an hs_err file. Cores are often due to some issue in the native code related to stream, but can also occur in the native code implementing the JVM itself.

To save the core data, first rename the core file, and create a pstack:
 # pstack core1 > pstack1.txt

Gather all information that might be of interest in a .tar.gz file, e.g.:
 # tar cf info_core1.tar /apps/mas/pstack1.txt /apps/mas/hs_err*
/apps/mas/log/* /apps/mas/cfg/* /apps/mas/etc/*
/etc/sfw/mobeon/MOBYmas/mas
 # gzip info_core1.tar

Submit a trouble report with the .tar.gz file attached, save the core file for later analysis by the appropriate designer. Again, be sure to include any non-standard native library/jar files (black builds) in use when performing debug activities.

The stack for the particular thread in which a crash has occurred usually contains the string "abort". If "abort" is found in a pstack, the stack of the crashing thread can be copy-pasted into the trouble report text to assist the TRB in identifying the part of the code which triggered a core dump. This information can also be deduced from the hs_err file (should there be one).

2.4 JVM GC and heap monitoring

2.4.1 General

Changes in the JVM GC parameters may become necessary e.g. due to code changes, running MAS on new architectures, resource pool resizing, traffic model changes and the execution of different VVA/VVAMCP package combinations. There is a large amount of documentation regarding JVM GC tuning, refer to e.g. [4] for an overview.

Should the JVM run out of Java heap during the testing, it will generate a heap dump (in /apps/mas) which is essential for later analysis, so make sure any such dumps are saved.

To monitor the effectiveness of a particular set of GC choices while running traffic, a visual tool such as VisualGC (included in JVMstat [5]) is highly useful. VisualGC can be run on the host running MAS while the graphical display is exported (e.g. export DISPLAY=10.20.30.40:0.34) to a remote x-server at 10.20.30.40 (run xhost + there). The "0.34" part can be found by using "who | grep <username>" at the x-server host. Set the paths required by VisualGC (i.e. to the JDK) and run "./visualgc <MAS_PID>".

Logging the garbage collection operations to a file can be useful while testing, e.g. to monitor the application stop times. Useful JVM flags include:

-Xloggc:/apps/mas/log/gc.log

-XX:+PrintGCDetails

-XX:+PrintGCTimeStamps

-XX:+PrintClassHistogram

-XX:+PrintGCApplicationConcurrentTime

-XX:+PrintGCApplicationStoppedTime

Use e.g. "tail –f gc.log | grep stopped" to get a view of the application thread stop times while observing the VisualGC output (if ApplicationStoppedTime is logged).

The standard tools included with the JDK are also useful for basic heap monitoring and heap dump analysis (e.g. jmap –histo, jmap –heap and jconsole).

Connecting to the JVM remotely using jconsole requires that the JVM has been started with the argument "+JMXremote". Jconsole also allows forcing the JVM to run GC at any time (although this does not guarantee a full GC).

For more thorough memory analysis activities, tools such as OptimizeIT can be used.

2.4.2 Finding potential GC issues

Extremely basic guidelines:

- Log the GC stop times as described in the previous section
- Consider running under JDK6.0 rather than 5.0 (for better heap dumps)
- Apply load
- Observe the application thread pauses in gc.log
    - Tail –f gc.log | grep "opped:"
- Should there be frequent stops >1s, there are issues.
    - Produce a heap dump, submit an error report. Save dump.
- If very large stops are observed, this indicates Java memory leak and/or a poor JVM parametrization for the load/architecture.
- If there are OutOfMemory log entries in mas.log or process.log, a memory leak or poor JVM parametrization is extremely likely.

2.5 Traffic Analysis

2.5.1 General

The Solaris snoop utility can be used to collect traffic, e.g. to an output file (-o). Various filters can be employed to reduce the amount of data in the snoop file, only including the traffic of interest (e.g. some particular backend or telephony protocol, UDP/TCP ports or target peer IP addresses). When snooping at high traffic load, it may be necessary to perform the snooping on a dedicated host (e.g. the secondary interface of a load generator host) by mirroring the MAS switch port to the snoop host port in the switch.

There are a number of other Solaris utilities which may also be of use in these activities, e.g. netstat (which shows network connection statistics) and pfiles (which shows all file descriptors, including network oriented, currently in use by the process).

2.5.2 Telephony traffic

Telephony traffic consists of the SIP signaling (normally inbound/outbound on UDP or TCP port 5060) and RTP/RTCP streams (normally inbound/outbound on UDP ports 23000 and upwards within the configured stream port pool range).

2.5.2.1 Capture

The Wireshark tool [6] can then be used to open and analyze collected snoop files (it can also be used for online analysis). Filters can also be applied offline in Wireshark. Rudimentary analysis of e.g. RTP streams can be performed by right clicking on a packet in a stream of interest, decoding it as RTP and then using statistics-RTP-stream analysis.

If all signaling and media traffic for a particular (full or partial) voice call has been captured in a snoop file, it is possible to decode (Statistics – VoIP calls), listen to media streams using various jitter buffer settings for decoding and view the SIP signaling.

For many purposes, capturing only outbound UDP and all SIP signaling (when snooping on a dedicated host where the MAS traffic is mirrored to a secondary interface, e.g. bge2) is a good way to help reduce the amount of traffic captured, while still being able to decode calls in this way:

 # snoop –d bge2 –o myTraffic.pkts port 5060 or udp and src <MAS_IP>

If the MAS connected to switch port x, connect the snoop host bge2 interface (typically, this is the third port on a quad port interface) to a free port on the switch, e.g. port y. Log into the Extreme Networks switch and show the port mirroring status using (the specifics may depend on the switch firmware version):

Switch # show mirroring

If no mirroring is currently set up, enable mirroring e.g. using these commands:

Switch # enable mirroring to port y

Switch # configure mirroring add port x

To disable the mirroring (if configured), use:

Switch # disable mirroring

Switch # configure mirroring delete port x

Bidirectional mirroring may also require that switch/host ports are set to half-duplex mode (use the Solaris ndd command to change this on a host). When snooped files are to be opened and analyzed in Wireshark, keeping the number of packets per snoop file below 100000 seems to be a good idea (e.g. make multiple snoop files or post-process captured data using "snoop –i" to reduce the amount of data before opening the file in Wireshark).

2.5.2.2 Telephony and RTP Stream Analysis

A simple option for checking the quality of single streams while the MAS is under load is to place a call to the MAS (which is under a background load from e.g. sipp channels or an MTG) using Kapanga, while capturing data on the local PC using Wireshark. After the call, stop capturing and decode VoIP. This also allows viewing the SIP signaling of each call graphically and with timing information.

The Wireshark tool can be used to analyze various quality metrics on individual RTP streams, mainly on the transport level. To analyze an individual stream, select a single packet belonging to the stream of interest, right-click and decode as RTP. Then use statistics-RTP-stream analysis to calculate e.g. packet delta, jitter and instantaneous bandwidth for the stream the packet belongs to.

The statistics for a particular stream can be exported to a .csv file for import into excel as a comma-separated file. In excel, it may be interesting e.g. to calculate the average (AVERAGE) delta between packets in a stream in order to assess any clock drift tendencies in the sender. The average delta should be close to the negotiated pTime for a particular call. Remember to remove the first packet of each play job from the calculation; these are usually the large deltas which still do not seem to introduce much/any jitter. The standard deviation of the delta (STDEV) is also interesting to calculate, this number should preferably be in the single digits.

A stream quality oriented tool, called Tracebuster [7], is available on a public laptop in the lab. It allows thorough online/offline analysis of captured network data, mainly with respect to telephony traffic (SIP and RTP) and the calculation of high level QoS metrics (MOS/R-values etc). When measuring directly on outbound traffic from the MAS, these figures should be virtually perfect according to the standards. The metrics and the use of this tool are described further in the documentation included with the tool.

To be able to open files created using the Sun snoop command, they must first be converted to standard pcap format, e.g. using Wireshark (open file, save in pcap format). For online analysis, connect the laptop to a mirrored switch port.

Short checklist for basic stream quality analysis

- Snoop all SIP and outbound UDP traffic to a file
- Open snoop file in Wireshark
- Decode VoIP calls with a jitter buffer of e.g. 50ms
- Listen to audio streams, check jitter/loss/reordering levels
- Check properties further for individual streams using excel import

- Open snoop files in Tracebuster
  - First convert the file to libpcap standard format

3 Test object Configuration

3.1 Standard Credentials

Intentionally left blank.

3.2 Basic Population

Intentionally left blank.

3.3 Deviating Configuration

JVM parameters can be modified in the /etc/sfw/mobeon/MOBYmas/mas file, some of them may also be set in the ~/etc/mas.conf file (~ denotes the MAS installation directory, /apps/mas by default).

Resource pools are mainly configured in mas.xml (e.g. LDAP and RTP port pool sizes) and in ComponentConfig.xml (e.g. Java thread pool sizes).

3.4 Checkpoints

Intentionally left blank.

3.5 Tips & Tricks

3.5.1 Capacity test

This cover some things to think of when doing capacity test on MAS.

3.5.1.1 Hammer

Make sure "Archive log msg" is activated, so that the log is stored to a file.

3.5.1.2 SIPp

You have to set a media port (-mp) that is different then 6000 if you start more then one (the SIPp uses port 6000 as default).

3.5.1.3 MVASSim

Use MVASSim to measure the response time from backend, i.e. LDAP and IMAP.

4 References

[1] Kapanga softphone
http://www.kapanga.org

[2] SIPp tool
http://sipp.sourceforge.net

[3] Developing and Tuning Applications on Ultrasparc T1 Chip Multithreading Systems
http://www.sun.com/blueprints/1205/819-5144.pdf

[4] Tuning Garbage Collection with the 5.0 Java Virtual Machine
http://java.sun.com/docs/hotspot/gc5.0/gc_tuning_5.html

[5] JVMstat package
http://java.sun.com/performance/jvmstat

[6] Wireshark

http://www.wireshark.org

[7] Touchstone Inc

http://www.touchstone-inc.com

5 Terminology

MAS	Media Access Server
JVM	Java Virtual Machine

6 Appendix