sysmocom

sysmocom - s.f.m.c. GmbH

Osmo-GSM-Tester Manual

by Neels Hofmeyr

The Asciidoc source code of this manual can be found at http://git.osmocom.org/osmo-gsm-manuals/

HISTORY			
NUMBER	DATE	DESCRIPTION	NAME
1	April 13, 2017	Initial version.	NH

Contents

1 WARNING: Work in Progress

NOTE: The osmo-gsm-tester is still in pre-alpha stage: some parts are still incomplete, and details will still change and move around.

2 Introduction with Examples

The osmo-gsm-tester is software to run automated tests of real GSM hardware, foremost to verify that ongoing Osmocom software development continues to work with various BTS models, while being flexibly configurable and extendable.

A *main unit* (general purpose computer) is connected via ethernet and/or USB to any number of BTS models and to any number of GSM modems via USB. The modems and BTS instances' RF transceivers are typically wired directly to each other via RF distribution chambers to bypass the air medium and avoid disturbing real production cellular networks. Furthermore, the setup may include adjustable RF attenuators to model various distances between modems and base stations.

The osmo-gsm-tester software runs on the main unit to orchestrate the various GSM hardware and run predefined test scripts. It typically receives binary packages from a jenkins build service. It then automatically configures and launches an Osmocom core network on the main unit and sets up and runs BTS models as well as modems to form a complete ad-hoc GSM network. On this setup, predefined test suites, combined with various scenario definitions, are run to verify stability of the system.

The osmo-gsm-tester is implemented in Python (version 3). It uses the ofono daemon to control the modems connected via USB. BTS software is either run directly on the main unit (e.g. for osmo-bts-trx, osmo-bts-octphy), run via SSH (e.g. for a sysmoBTS) or assumed to run on a connected BTS model (e.g. for ip.access nanoBTS).

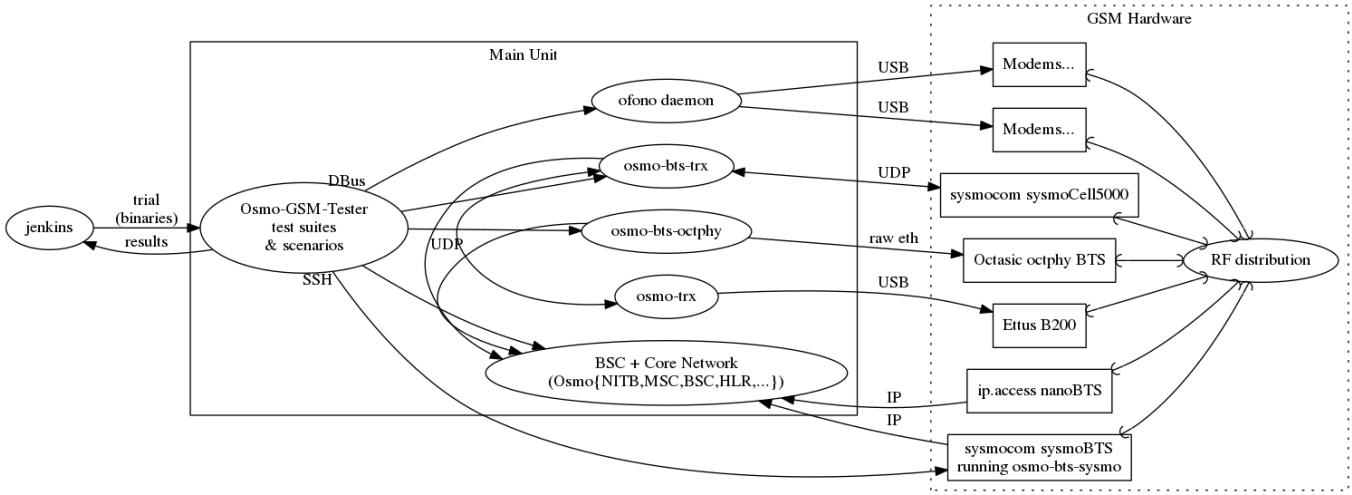

Figure 1: Typical osmo-gsm-tester setup

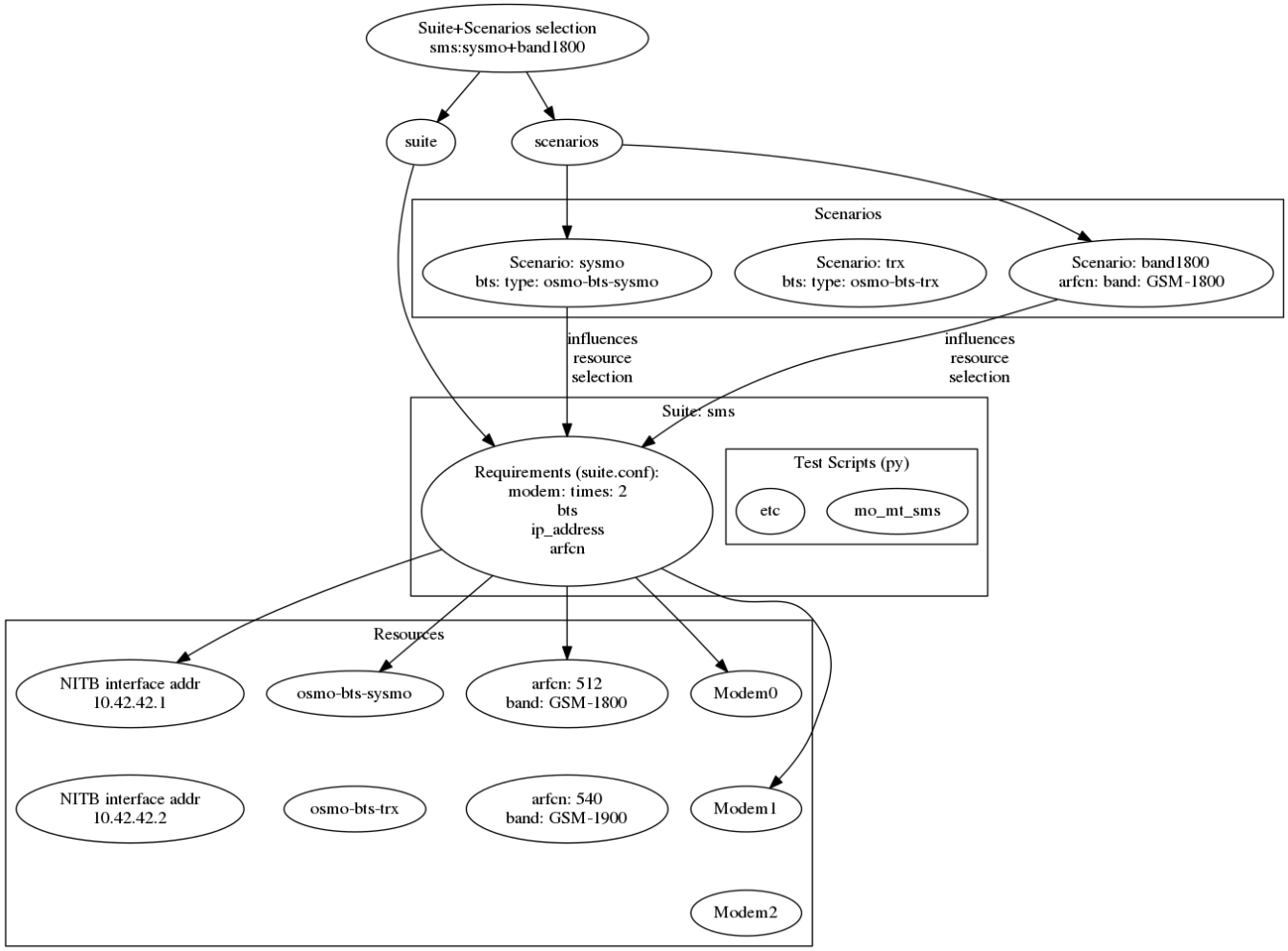

Figure 2: Example of how to select resources and configurations: scenarios may pick specific resources (here BTS and ARFCN), remaining requirements are picked as available (here two modems and a NITB interface)

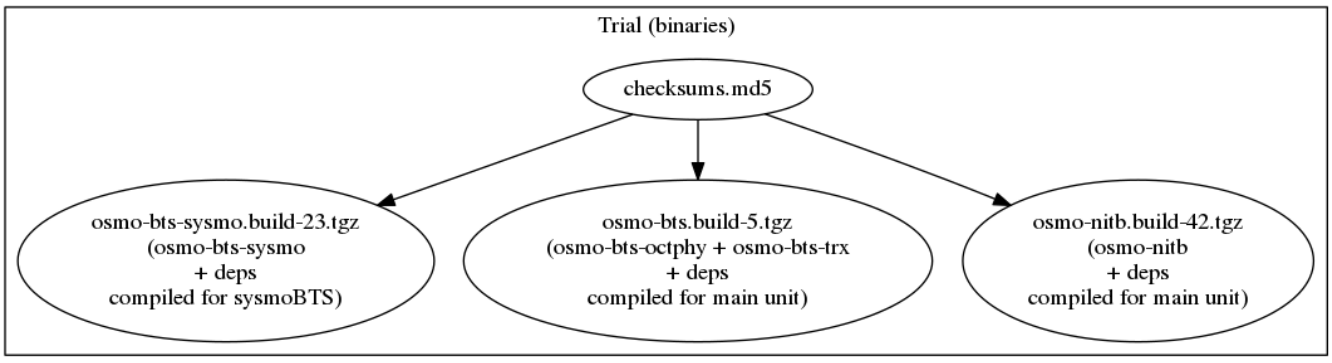

Figure 3: Example of a "trial" containing binaries built by a jenkins

2.1 Typical Test Script

A typical single test script (part of a suite) may look like this:


```
#!/usr/bin/env python3
from osmo_gsm_tester.testenv import *

hlr = suite.hlr()
bts = suite.bts()
mgcpgw = suite.mgcpgw(bts_ip=bts.remote_addr())
msc = suite.msc(hlr, mgcpgw)
bsc = suite.bsc(msc)
stp = suite.stp()
ms_mo = suite.modem()
ms_mt = suite.modem()

hlr.start()
stp.start()
msc.start()
mgcpgw.start()

bsc.bts_add(bts)
bsc.start()

bts.start()

hlr.subscriber_add(ms_mo)
hlr.subscriber_add(ms_mt)

ms_mo.connect(msc.mcc_mnc())
ms_mt.connect(msc.mcc_mnc())

ms_mo.log_info()
ms_mt.log_info()

print('waiting for modems to attach...')
wait(ms_mo.is_connected, msc.mcc_mnc())
wait(ms_mt.is_connected, msc.mcc_mnc())
wait(msc.subscriber_attached, ms_mo, ms_mt)

sms = ms_mo.sms_send(ms_mt)
wait(ms_mt.sms_was_received, sms)
```

2.2 Resource Resolution

- A global configuration *resources.conf* defines which hardware is connected to the osmo-gsm-tester main unit.
- Each suite contains a number of test scripts. The amount of resources a test may use is defined by the test suite's *suite.conf*.
- Which specific modems, BTS models, NITB IP addresses etc. are made available to a test run is typically determined by *suite.conf* and a combination of scenario configurations — or picked automatically if not.

2.3 Typical *resources.conf*

A global configuration of hardware may look like below; for details, see Section 5.4.

```
ip_address:
- addr: 10.42.42.2
- addr: 10.42.42.3
- addr: 10.42.42.4
- addr: 10.42.42.5
- addr: 10.42.42.6
```



```
bts:
- label: sysmoBTS 1002
  type: osmo-bts-sysmo
  ipa_unit_id: 1
  addr: 10.42.42.114
  band: GSM-1800
  ciphers:
  - a5_0
  - a5_1
  - a5_3

- label: Ettus B200
  type: osmo-bts-trx
  ipa_unit_id: 6
  addr: 10.42.42.50
  band: GSM-1800
  launch_trx: true
  ciphers:
  - a5_0
  - a5_1

- label: sysmoCell 5000
  type: osmo-bts-trx
  ipa_unit_id: 7
  addr: 10.42.42.51
  band: GSM-1800
  trx_remote_ip: 10.42.42.112
  ciphers:
  - a5_0
  - a5_1

- label: OCTBTS 3500
  type: osmo-bts-octphy
  ipa_unit_id: 8
  addr: 10.42.42.52
  band: GSM-1800
  trx_list:
  - hw_addr: 00:0c:90:2e:80:1e
    net_device: eth1
  - hw_addr: 00:0c:90:2e:87:52
    net_device: eth1

arfcn:
  - arfcn: 512
    band: GSM-1800
  - arfcn: 514
    band: GSM-1800
  - arfcn: 516
    band: GSM-1800
  - arfcn: 546
    band: GSM-1900
  - arfcn: 548
    band: GSM-1900

modem:
- label: sierra_1
  path: '/sierra_1'
  imsi: '901700000009031'
  ki: '80A37E6FDEA931EAC92FFA5F671EFEAD'
  auth_algo: 'xor'
  ciphers:
  - a5_0
```



```
  - a5_1
  features:
  - 'sms'
  - 'voice'

- label: gobi_0
  path: '/gobi_0'
  imsi: '901700000009030'
  ki: 'BB70807226393CDBAC8DD3439FF54252'
  auth_algo: 'xor'
  ciphers:
  - a5_0
  - a5_1
  features:
  - 'sms'
```

2.4 Typical *suites/*/suite.conf*

The configuration that reserves a number of resources for a test suite may look like this:

```
resources:
  ip_address:
  - times: 1
  bts:
  - times: 1
  modem:
  - times: 2
    features:
    - sms
```

It may also request e.g. specific BTS models, but this is typically left to scenario configurations.

2.5 Typical *scenarios/*.conf*

For a suite as above run as-is, any available resources are picked. This may be combined with any number of scenario definitions to constrain which specific resources should be used, e.g.:

```
resources:
  bts:
  - type: osmo-bts-sysmo
```

Which *ip_address* or *modem* is used in particular doesn't really matter, so it can be left up to the osmo-gsm-tester to pick these automatically.

Any number of such scenario configurations can be combined in the form <*suite_name*>:<*scenario*>+<*scenario*>+..., e.g. *my_suite:sysmo+tch_f+amr*.

2.6 Typical Invocations

Each invocation of osmo-gsm-tester deploys a set of pre-compiled binaries for the Osmocom core network as well as for the Osmocom based BTS models. To create such a set of binaries, see Section 6.

Examples for launching test trials:

- Run the default suites (see Section 5.5) on a given set of binaries:

```
osmo-gsm-tester.py path/to/my-trial
```


- Run an explicit choice of *suite:scenario* combinations:

```
osmo-gsm-tester.py path/to/my-trial -s sms:sysmo -s sms:trx -s sms:nanobts
```

- Run one *suite:scenario* combination, setting log level to *debug* and enabling logging of full python tracebacks, and also only run just the *mo_mt_sms.py* test from the suite, e.g. to investigate a test failure:

```
osmo-gsm-tester.py path/to/my-trial -s sms:sysmo -l dbg -T -t mo_mt
```

A test script may also be run step-by-step in a python debugger, see Section 8.

2.7 Resource Reservation for Concurrent Trials

While a test suite runs, the used resources are noted in a global state directory in a reserved-resources file. This way, any number of trials may be run consecutively without resource conflicts. Any test trial will only use resources that are currently not reserved by any other test suite. The reservation state is human readable.

The global state directory is protected by a file lock to allow access by separate processes.

Also, the binaries from a trial are never installed system-wide, but are run with a specific *LD_LIBRARY_PATH* pointing at the trial's *inst*, so that several trials can run consecutively without conflicting binary versions. For some specific binaries which require extra permissions (such as osmo-bts-octphy requiring *CAP_NET_RAW*), *patchelf* program is used to modify the binary *RPATH* field instead because the OS dynamic linker skips *LD_LIBRARY_PATH* for binaries with special permissions.

Once a test suite run is complete, all its reserved resources are torn down (if the test scripts have not done so already), and the reservations are released automatically.

If required resources are unavailable, the test trial fails. For consecutive test trials, a test run needs to either wait for resources to become available, or test suites need to be scheduled to make sense. (← **TODO**)

3 Installation on Main Unit

The main unit is a general purpose computer that orchestrates the tests. It runs the core network components, controls the modems and so on. This can be anything from a dedicated production rack unit to your laptop at home.

This manual will assume that tests are run from a jenkins build slave, by a user named *jenkins* that belong to group *osmo-gsm-tester*. The user configuration for manual test runs and/or a different user name is identical, simply replace the user name or group.

3.1 Osmo-gsm-tester Dependencies

On a Debian/Ubuntu based system, these commands install the packages needed to run the osmo-gsm-tester.py code, i.e. install these on your main unit:

```
apt-get install \
  dbus \
  tcpdump \
  sqlite3 \
  python3 \
  python3-yaml \
  python3-mako \
  python3-gi \
  ofono \
  patchelf \
  sudo \
  libcap2-bin \
  python3-pip
pip3 install pydbus
pip3 install git+git://github.com/podshumok/python-smpplib.git
```


Important
ofono may need to be installed from source to contain the most recent fixes needed to operate your modems. This depends on the modem hardware used and the tests run. Please see Section 4.2.

To run osmo-bts-trx with a USRP attached, you may need to install a UHD driver. Please refer to http://osmocom.org/projects/-osmotrx/wiki/OsmoTRX#UHD for details; the following is an example for the B200 family USRP devices:

```
apt-get install libuhd-dev uhd-host
/usr/lib/uhd/utils/uhd_images_downloader.py
```

3.1.1 Osmocom Build Dependencies

Each of the jenkins builds requires individual dependencies. This is generally the same as for building the software outside of osmo-gsm-tester and will not be detailed here. For the Osmocom projects, refer to http://osmocom.org/projects/cellular-infrastructure/wiki/Build_from_Source . Be aware of specific requirements for BTS hardware: for example, the osmo-bts-sysmo build needs the sysmoBTS SDK installed on the build slave, which should match the installed sysmoBTS firmware.

3.2 Jenkins Build and Run Slave

3.2.1 Create *jenkins* User on Main Unit

On the main unit, create a jenkins user:

```
useradd -m jenkins
```

3.2.2 Install Java on Main Unit

To be able to launch the Jenkins build slave, a Java RE must be available on the main unit. For example:

```
apt-get install default-jdk
```

3.2.3 Allow SSH Access from Jenkins Master

Create an SSH keypair to be used for login on the osmo-gsm-tester. This may be entered on the jenkins web UI; alternatively, use the jenkins server's shell:

Login on the main jenkins server shell and create an SSH keypair, for example:

```
# su jenkins
$ mkdir -p /usr/local/jenkins/keys
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/jenkins/.ssh/id_rsa): /usr/local/jenkins/keys/ ↩
    osmo-gsm-tester-rnd
Enter passphrase (empty for no passphrase): <enter a passphrase>
Enter same passphrase again: <enter a passphrase>
Your identification has been saved in /usr/local/jenkins/keys/osmo-gsm-tester-rnd
Your public key has been saved in /usr/local/jenkins/keys/osmo-gsm-tester-rnd.pub.
The key fingerprint is:
...
```

Copy the public key to the main unit, e.g. copy-paste:


```
cat /usr/local/jenkins/keys/osmo-gsm-tester-rnd.pub
# copy this public key
```

On the main unit:

```
mkdir ~jenkins/.ssh
cat > ~jenkins/.ssh/authorized_keys
# paste above public key and hit Ctrl-D
chown -R jenkins: ~jenkins/.ssh
```

Make sure that the user running the jenkins master accepts the main unit's host identification. There must be an actual RSA host key available in the known_hosts file for the jenkins master to be able to log in. Simply calling ssh and accepting the host key as usual is not enough. Jenkins may continue to say "Host key verification failed".

To place an RSA host key in the jenkins' known_hosts file, you may do:

On the Jenkins master:

```
main_unit_ip=10.9.8.7
ssh-keyscan -H $main_unit_ip >> ~jenkins/.ssh/known_hosts
chown jenkins: ~jenkins/.ssh/known_hosts
```

Verify that the jenkins user on the Jenkins master has SSH access to the main unit:

```
su jenkins
main_unit_ip=10.9.8.7
ssh -i /usr/local/jenkins/keys/osmo-gsm-tester-rnd jenkins@$main_unit_ip
exit
```

3.2.4 Add Jenkins Slave

In the jenkins web UI, add a new build slave for the osmo-gsm-tester:

- *Manage Jenkins*
  - *Manage Nodes*
    - *New Node*
      - Enter a node name, e.g. "osmo-gsm-tester-1"
        (the "-1" is just some identification in case you'd like to add another setup later).
      - *Permanent Agent*

Configure the node as:

- *# of executors*: 1
- *Remote root directory*: "/home/jenkins"
- *Labels*: "osmo-gsm-tester"
  (This is a general label common to all osmo-gsm-tester build slaves you may set up in the future.)
- *Usage*: *Only build jobs with label expressions matching this node*
- *Launch method*: *Launch slave agents via SSH*
  - *Host*: your main unit's IP address
  - *Credentials*: choose *Add* / *Jenkins*
    - *Domain*: *Global credentials (unrestricted)*
    - *Kind*: *SSH Username with private key*

    * *Scope*: *Global*
    * *Username*: "jenkins"
      (as created on the main unit above)
    * *Private Key*: *From a file on Jenkins master*
      · *File*: "/usr/local/jenkins/keys/osmo-gsm-tester-rnd"
    * *Passphrase*: enter same passphrase as above
    * *ID*: "osmo-gsm-tester-1"
    * *Name*: "jenkins for SSH to osmo-gsm-tester-1"

The build slave should be able to start now.

3.2.5 Add Build Jobs

There are various jenkins-build-* scripts in osmo-gsm-tester/contrib/, which can be called as jenkins build jobs to build and bundle binaries as artifacts, to be run on the osmo-gsm-tester main unit and/or BTS hardware.

Be aware of the dependencies, as hinted at in Section 3.1.1.

While the various binaries could technically be built on the osmo-gsm-tester main unit, it is recommended to use a separate build slave, to take load off of the main unit.

On your jenkins master, set up build jobs to call these scripts — typically one build job per script. Look in contrib/ and create one build job for each of the BTS types you would like to test, as well as one for the *build-osmo-nitb*.

These are generic steps to configure a jenkins build job for each of these build scripts, by example of the jenkins-build-osmo-nitb.sh script; all that differs to the other scripts is the "osmo-nitb" part:

- *Project name*: "osmo-gsm-tester_build-osmo-nitb"
  (Replace *osmo-nitb* according to which build script this is for)
- *Discard old builds*
  Configure this to taste, for example:
  - *Max # of build to keep*: "20"
- *Restrict where this project can be run*: Choose a build slave label that matches the main unit's architecture and distribution, typically a Debian system, e.g.: "linux_amd64_debian8"
- *Source Code Management*:
  - *Git*
    * *Repository URL*: "git://git.osmocom.org/osmo-gsm-tester"
    * *Branch Specifier*: "*/master"
    * *Additional Behaviors*
      · *Check out to a sub-directory*: "osmo-gsm-tester"
- *Build Triggers*
  The decision on when to build is complex. Here are some examples:
  - Once per day:
    Build periodically: "H H * * *"
  - For the Osmocom project, the purpose is to verify our software changes. Hence we would like to test every time our code has changed:
    * We could add various git repositories to watch, and enable *Poll SCM*.
    * On jenkins.osmocom.org, we have various jobs that build the master branches of their respective git repositories when a new change was merged. Here, we can thus trigger e.g. an osmo-nitb build for osmo-gsm-tester everytime the master build has run:
      Build after other projects are built: "OpenBSC"

    * Note that most of the Osmocom projects also need to be re-tested when their dependencies like libosmo* have changed. Triggering on all those changes typically causes more jenkins runs than necessary: for example, it rebuilds once per each dependency that has rebuilt due to one libosmocore change. There is so far no trivial way known to avoid this. It is indeed safest to rebuild more often.

- *Build*
  - *Execute Shell*

```
#!/bin/sh
set -e -x
./osmo-gsm-tester/contrib/jenkins-build-osmo-nitb.sh
```

    (Replace *osmo-nitb* according to which build script this is for)

- *Post-build Actions*
  - *Archive the artifacts*: "*.tgz, *.md5"
    (This step is important to be able to use the built binaries in the run job below.)

Tip
When you've created one build job, it is convenient to create further build jobs by copying the first and, e.g., simply replacing all "osmo-nitb" with "osmo-bts-trx".

3.2.6 Add Run Job

This is the jenkins job that runs the tests on the GSM hardware:

- It sources the artifacts from jenkins' build jobs.
- It runs on the osmo-gsm-tester main unit.

Here is the configuration for the run job:

- *Project name*: "osmo-gsm-tester_run"
- *Discard old builds*
  Configure this to taste, for example:
  - *Max # of build to keep*: "20"
- *Restrict where this project can be run*: "osmo-gsm-tester"
  (to match the *Label* configured in Section 3.2.4).
- *Source Code Management*:
  - *Git*
    * *Repository URL*: "git://git.osmocom.org/osmo-gsm-tester"
    * *Branch Specifier*: "*/master"
    * *Additional Behaviors*
      · *Check out to a sub-directory*: "osmo-gsm-tester"
      · *Clean before checkout*
- *Build Triggers*
  The decision on when to build is complex. For this run job, it is suggested to rebuild:

  - after each of above build jobs that produced new artifacts:
    Build after other projects are built: "osmo-gsm-tester_build-osmo-nitb, osmo-gsm-tester_build-osmo-bts-sysmo, osmo-gsm-tester_build-osmo-bts-trx"
    (Add each build job name you configured above)
  - as well as once per day:
    Build periodically: "H H * * *"
  - and, in addition, whenever the osmo-gsm-tester scripts have been modified:
    Poll SCM: "H/5 * * * *"
    (i.e. look every five minutes whether the upstream git has changed)

- *Build*
  - Copy artifacts from each build job you have set up:
    * *Copy artifacts from another project*
      · *Project name*: "osmo-gsm-tester_build-osmo-nitb"
      · *Which build*: *Latest successful build*
      · enable *Stable build only*
      · *Artifacts to copy*: "*.tgz, *.md5"
    * Add a separate similar *Copy artifacts...* section for each build job you have set up.
  - *Execute Shell*

```
#!/bin/sh
set -e -x

# debug: provoke a failure
#export OSMO_GSM_TESTER_OPTS="-s debug -t fail"

PATH="$PWD/osmo-gsm-tester/src:$PATH" \
  ./osmo-gsm-tester/contrib/jenkins-run.sh
```

    Details:
    * The *jenkins-run.sh* script assumes to find the *osmo-gsm-tester.py* in the *$PATH*. To use the most recent osmo-gsm-tester code here, we direct *$PATH* to the actual workspace checkout. This could also run from a sytem wide install, in which case you could omit the explicit PATH to "$PWD/osmo-gsm-tester/src".
    * This assumes that there are configuration files for osmo-gsm-tester placed on the system (see Section 5.1).
    * If you'd like to check the behavior of test failures, you can uncomment the line below "# debug" to produce a build failure on every run. Note that this test typically produces a quite empty run result, since it launches no NITB nor BTS.

- *Post-build Actions*
  - *Archive the artifacts*
    * *Files to archive*: "*-run.tgz, *-bin.tgz"
      This stores the complete test report with config files, logs, stdout/stderr output, pcaps as well as the binaries used for the test run in artifacts. This allows analysis of older builds, instead of only the most recent build (which cleans up the jenkins workspace every time). The *trial-N-run.tgz* and *trial-N-bin.tgz* archives are produced by the *jenkins-run.sh* script, both for successful and failing runs.

3.3 Install osmo-gsm-tester on Main Unit

This assumes you have already created the jenkins user (see Section 3.2).

3.3.1 User Permissions

On the main unit, create a group for all users that should be allowed to use the osmo-gsm-tester, and add users (here *jenkins*) to this group.

```
groupadd osmo-gsm-tester
gpasswd -a jenkins osmo-gsm-tester
```

Note
you may also need to add users to the *usrp* group, see Section 3.3.1.6.

A user added to a group needs to re-login for the group permissions to take effect.

This group needs the following permissions:

3.3.1.1 Paths

Assuming that you are using the example config, prepare a system wide state location in */var/tmp*:

```
mkdir -p /var/tmp/osmo-gsm-tester/state
chown -R :osmo-gsm-tester /var/tmp/osmo-gsm-tester
chmod -R g+rwxs /var/tmp/osmo-gsm-tester
setfacl -d -m group:osmo-gsm-tester:rwx /var/tmp/osmo-gsm-tester/state
```

Important
the state directory needs to be shared between all users potentially running the osmo-gsm-tester to resolve resource allocations. Above *setfacl* command sets the access control to keep all created files group writable.

With the jenkins build as described here, the trials will live in the build slave's workspace. Other modes of operation (a daemon scheduling concurrent runs, **TODO**) may use a system wide directory to manage trials to run:

```
mkdir -p /var/tmp/osmo-gsm-tester/trials
chown -R :osmo-gsm-tester /var/tmp/osmo-gsm-tester
chmod -R g+rwxs /var/tmp/osmo-gsm-tester
```

3.3.1.2 Allow DBus Access to ofono

Put a DBus configuration file in place that allows the *osmo-gsm-tester* group to access the org.ofono DBus path:

```
cat > /etc/dbus-1/system.d/osmo-gsm-tester.conf <<END
<!-- Additional rules for the osmo-gsm-tester to access org.ofono from user
     land -->

<!DOCTYPE busconfig PUBLIC "-//freedesktop//DTD D-BUS Bus Configuration 1.0//EN"
 "http://www.freedesktop.org/standards/dbus/1.0/busconfig.dtd">
<busconfig>

  <policy group="osmo-gsm-tester">
    <allow send_destination="org.ofono"/>
  </policy>

</busconfig>
END
```

(No restart of dbus nor ofono necessary.)

3.3.1.3 Capture Packets

In order to allow collecting pcap traces of the network communication for later reference, allow the osmo-gsm-tester group to capture packets using the *tcpdump* program:

```
chgrp osmo-gsm-tester /usr/sbin/tcpdump
chmod 750 /usr/sbin/tcpdump
setcap cap_net_raw,cap_net_admin=eip /usr/sbin/tcpdump
```

Put *tcpdump* in the *$PATH* — assuming that *tcpdump* is available for root:

```
ln -s `which tcpdump` /usr/local/bin/tcpdump
```

Tip
Why a symlink in */usr/local/bin*? On Debian, *tcpdump* lives in */usr/sbin*, which is not part of the *$PATH* for non-root users. To avoid hardcoding non-portable paths in the osmo-gsm-tester source, *tcpdump* must be available in the *$PATH*. There are various trivial ways to modify *$PATH* for login shells, but the jenkins build slave typically runs in a **non-login** shell; modifying non-login shell enviroments is not trivially possible without also interfering with files installed from debian packages. Probably the easiest way to allow all users and all shells to find the *tcpdump* binary is to actually place a symbolic link in a directory that is already part of the non-login shell's *$PATH*. Above example places such in */usr/local/bin*.

Verify that a non-login shell can find *tcpdump*:

```
su jenkins -c 'which tcpdump'
# should print: "/usr/local/bin/tcpdump"
```

Warning
When logged in via SSH on your main unit, running *tcpdump* to capture packets may result in a feedback loop: SSH activity to send tcpdump's output to your terminal is in turn is picked up in the tcpdump trace, and so forth. When testing *tcpdump* access, make sure to have proper filter expressions in place.

3.3.1.4 Allow Core Files

In case a binary run for the test crashes, a core file of the crash should be written. This requires a limit rule. Create a file with the required rule:

```
sudo -s
echo "@osmo-gsm-tester - core unlimited" > /etc/security/limits.d/osmo-gsm-tester_allow- ↩
    core.conf
```

Re-login the user to make these changes take effect.

Set the **kernel.core_pattern** sysctl to **core** (usually the default). For each binary run by osmo-gsm-tester, a core file will then appear in the same dir that contains stdout and stderr for that process (because this dir is set as CWD).

```
sysctl -w kernel.core_pattern=core
```

3.3.1.5 Allow Realtime Priority

Certain binaries should be run with real-time priority, like *osmo-bts-trx*. Add this permission on the main unit:

```
sudo -s
echo "@osmo-gsm-tester - rtprio 99" > /etc/security/limits.d/osmo-gsm-tester_allow-rtprio. ↩
    conf
```

Re-login the user to make these changes take effect.

3.3.1.6 UHD

Grant permission to use the UHD driver to run USRP devices for osmo-bts-trx, by adding the jenkins user to the *usrp* group:

```
gpasswd -a jenkins usrp
```

3.3.1.7 Allow CAP_NET_RAW capability

Certain binaries require *CAP_NET_RAW* to be set, like *osmo-bts-octphy* as it uses a *AF_PACKET* socket.

To be able to set the following capability without being root, osmo-gsm-tester uses sudo to gain permissions to set the capability.

This is the script that osmo-gsm-tester expects on the main unit:

```
echo /usr/local/bin/osmo-gsm-tester_setcap_net_raw.sh <<EOF
#!/bin/bash
/sbin/setcap cap_net_raw+ep $1
EOF
chmod +x /usr/local/bin/osmo-gsm-tester_setcap_net_raw.sh
```

Now, again on the main unit, we need to provide sudo access to this script for osmo-gsm-tester:

```
echo "%osmo-gsm-tester ALL=(root) NOPASSWD: /usr/local/bin/osmo-gsm-tester_setcap_net_raw. ↩
    sh" > /etc/sudoers.d/osmo-gsm-tester_setcap_net_raw
chmod 0440 /etc/sudoers.d/osmo-gsm-tester_setcap_net_raw
```

The script file name *osmo-gsm-tester_setcap_net_raw.sh* is important, as osmo-gsm-tester expects to find a script with this name in *$PATH* at run time.

3.3.2 Log Rotation

To avoid clogging up /var/log, it makes sense to choose a sane maximum log size:

```
echo maxsize 10M > /etc/logrotate.d/maxsize
```

3.3.3 Install Scripts

Important
When using the jenkins build slave as configured above, **there is no need to install the osmo-gsm-tester sources on the main unit**. The jenkins job will do so implicitly by checking out the latest osmo-gsm-tester sources in the workspace for every run. If you're using only the jenkins build slave, you may skip this section.

If you prefer to use a fixed installation of the osmo-gsm-tester sources instead of the jenkins workspace, you can:

1. From the run job configured above, remove the line that says

   ```
   PATH="$PWD/osmo-gsm-tester/src:$PATH" \
   ```

   so that this uses a system wide installation instead.

2. Install the sources e.g. in */usr/local/src* as indicated below.

On the main unit, to install the latest in */usr/local/src*:


```
apt-get install git
mkdir -p /usr/local/src
cd /usr/local/src
git clone git://git.osmocom.org/osmo-gsm-tester
```

To allow all users to run *osmo-gsm-tester.py*, from login as well as non-login shells, the easiest solution is to place a symlink in */usr/local/bin*:

```
ln -s /usr/local/src/osmo-gsm-tester/src/osmo-gsm-tester.py /usr/local/bin/
```

(See also the tip in Section 3.3.1.3 for a more detailed explanation.)

The example configuration provided in the source is suitable for running as-is, **if** your hardware setup matches (you could technically use that directly by a symlink e.g. from */usr/local/etc/osmo-gsm-tester* to the *example* dir). If in doubt, rather copy the example, point *paths.conf* at the *suites* dir, and adjust your own configuration as needed. For example:

```
cd /etc
cp -R /usr/local/src/osmo-gsm-tester/example osmo-gsm-tester
sed -i 's#\.\./suites#/usr/local/src/osmo-gsm-tester/suites#' osmo-gsm-tester/paths.conf
```

Note
The configuration will be looked up in various places, see Section 5.1.

4 Hardware Choice and Configuration

4.1 SysmoBTS

To use the SysmoBTS in the osmo-gsm-tester, the following systemd services must be disabled:

```
systemctl mask osmo-nitb osmo-bts-sysmo osmo-pcu sysmobts-mgr
```

This stops the stock setup keeping the BTS in operation and hence allows the osmo-gsm-tester to install and launch its own versions of the SysmoBTS software.

4.1.1 IP Address

To ensure that the SysmoBTS is always reachable at a fixed known IP address, configure the eth0 to use a static IP address:

Adjust */etc/network/interfaces* and replace the line

```
iface eth0 inet dhcp
```

with

```
iface eth0 inet static
  address 10.42.42.114
  netmask 255.255.255.0
  gateway 10.42.42.1
```

You may set the name server in */etc/resolve.conf* (most likely to the IP of the gateway), but this is not really needed by the osmo-gsm-tester.

4.1.2 Allow Core Files

In case a binary run for the test crashes, a core file of the crash should be written. This requires a limits rule. Append a line to /etc/limits like:

```
ssh root@10.42.42.114
echo "* C16384" >> /etc/limits
```

4.1.3 Reboot

Reboot the BTS and make sure that the IP address for eth0 is now indeed 10.42.42.114, and that no osmo* programs are running.

```
ip a
ps w | grep osmo
```

4.1.4 SSH Access

Make sure that the jenkins user on the main unit is able to login on the sysmoBTS, possibly erasing outdated host keys after a new rootfs was loaded:

On the main unit, for example do:

```
su - jenkins
ssh root@10.42.42.114
```

Fix any problems until you get a login on the sysmoBTS.

4.2 Modems

TODO: describe modem choices and how to run ofono

4.3 osmo-bts-trx

TODO: describe B200 family

5 Configuration

5.1 Config Paths

The osmo-gsm-tester looks for configuration files in various standard directories in this order:

- *$HOME/.config/osmo-gsm-tester/*
- */usr/local/etc/osmo-gsm-tester/*
- */etc/osmo-gsm-tester/*

The config location can also be set by an environment variable *$OSMO_GSM_TESTER_CONF*, which then overrides the above locations.

The osmo-gsm-tester expects to find the following configuration files in a configuration directory:

- *paths.conf*

- *resources.conf*
- *default-suites.conf* (optional)
- *defaults.conf* (optional)

These are described in detail in the following sections.

5.2 Format: YAML, and its Drawbacks

The general configuration format used is YAML. The stock python YAML parser does have several drawbacks: too many complex possibilities and alternative ways of formatting a configuration, but at the time of writing seems to be the only widely used configuration format that offers a simple and human readable formatting as well as nested structuring. It is recommended to use only the exact YAML subset seen in this manual in case the osmo-gsm-tester should move to a less bloated parser in the future.

Careful: if a configuration item consists of digits and starts with a zero, you need to quote it, or it may be interpreted as an octal notation integer! Please avoid using the octal notation on purpose, it is not provided intentionally.

5.3 *paths.conf*

The *paths.conf* file defines where to store the global state (of reserved resources) and where to find suite and scenario definitions.

Any relative paths found in a *paths.conf* file are interpreted as relative to the directory of that *paths.conf* file.

Example:

```
state_dir: '/var/tmp/osmo-gsm-tester/state'
suites_dir: '/usr/local/src/osmo-gsm-tester/suites'
scenarios_dir: './scenarios'
```

5.3.1 *state_dir*

It contains global or system-wide state for osmo-gsm-tester. In a typical state dir you can find the following files:

last_used_msisdn.state
: Contains last used msisdn number, which is automatically increased every time osmo-gsm-tester needs to assign a new subscriber in a test.

lock
: Lock file used to implement a mutual exclusion zone around the *reserved_resources.state* file.

reserved_resources.state
: File containing a set of reserved resources by any number of osmo-gsm-tester instances. Each osmo-gsm-tester instance is responsible to clear its resources from the list once it is done using them and are no longer reserved.

If you would like to set up several separate configurations (not typical), note that the *state_dir* is used to reserve resources, which only works when all configurations that share resources also use the same *state_dir*.

This way, several concurrent users of osmo-gsm-tester (ie. several osmo-gsm-tester processes running in parallel) can run without interfering with each other (e.g. using same ARFCN, same IP or same ofono modem path).

5.3.2 *suites_dir*

Suites contain a set of tests which are designed to be run together to test a set of features given a specific set of resources. As a result, resources are allocated per suite and not per test.

Tests for a given suite are located in the form of *.py* python scripts in the same directory where the *suite.conf* lays.

5.3.3 *scenarios_dir*

This dir contains scenario configuration files.

Scenarios define constraints to serve the resource requests of a *suite.conf*, to select specific resources from the general resource pool specified in *resources.conf*.

All *times* attributes are expanded before matching. For example, if a *suite.conf* requests two BTS, we may enforce that both BTS should be of type *osmo-bts-sysmo* in these ways:

```
resources:
  bts:
  - type: osmo-bts-sysmo
  - type: osmo-bts-sysmo
```

or alternatively,

```
resources:
  bts:
  - times: 2
    type: osmo-bts-sysmo
```

If only one resource is specified in the scenario, then the resource allocator assumes the restriction is to be applied to the first resource and that remaining resources have no restrictions to be taken into consideration.

To apply restrictions only on the second resource, the first element can be left emtpy, like:

```
resources:
  bts:
  - {}
  - type: osmo-bts-sysmo
```

On the *osmo_gsm_tester.py* command line and the *default_suites.conf*, any number of such scenario configurations can be combined in the form:

```
<suite_name>:<scenario>[+<scenario>[+...]]
```

e.g.

```
my_suite:sysmo+tch_f+amr
```

5.4 *resources.conf*

The *resources.conf* file defines which hardware is connected to the main unit, as well as which limited configuration items (like IP addresses or ARFCNs) should be used.

These resources are allocated dynamically and are not configured explicitly:

- MSISDN: phone numbers are dealt out to test scripts in sequence on request.

A *resources.conf* is structured as a list of items for each resource type, where each item has one or more settings — for an example, see Section 2.3.

These kinds of resource are known:

ip_address
: List of IP addresses to run osmo-nitb instances on. The main unit typically has a limited number of such IP addresses configured, which the connected BTS models can see on their network.

  addr
  : IPv4 address of the local interface.

bts

List of available BTS hardware.

label

human readable label for your own reference

type

which way to launch this BTS, one of

- *osmo-bts-sysmo*
- *osmo-bts-trx*
- *osmo-bts-octphy*
- *ipa-nanobts*

ipa_unit_id

ip.access unit id to be used by the BTS, written into BTS and BSC config.

addr

Remote IP address of the BTS for BTS like sysmoBTS, and local IP address to bind to for locally run BTS such as osmo-bts-trx.

band

GSM band that this BTS shoud use (**TODO**: allow multiple bands). One of:

- *GSM-1800*
- *GSM-1900*
- (**TODO**: more bands)

trx_list

Specific TRX configurations for this BTS. There should be as many of these as the BTS has TRXes. (**TODO**: a way to define >1 TRX without special configuration for them.)

hw_addr

Hardware (MAC) address of the TRX in the form of *11:22:33:44:55:66*, only used for osmo-bts-octphy. (**TODO**: and nanobts??)

net_device

Local network device to reach the TRX's *hw_addr* at, only used for osmo-bts-octphy. Example: *eth0*.

nominal_power

Nominal power to be used by the TRX.

max_power_red

Max power reduction to apply to the nominal power of the TRX.

arfcn

List of ARFCNs to use for running BTSes, which defines the actual RF frequency bands used.

arfcn

ARFCN number, see e.g. https://en.wikipedia.org/wiki/Absolute_radio-frequency_channel_number (note that the resource type *arfcn* contains an item trait also named *arfcn*).

band

GSM band name to use this ARFCN for, same as for *bts:band* above.

modem

List of modems reachable via ofono and information on the inserted SIM card. (Note: the MSISDN is allocated dynamically in test scripts).

label

Human readable label for your own reference, which also appears in logs.

path

Ofono's path for this modem, like */modemkind_99*.

imsi

IMSI of the inserted SIM card, like *"123456789012345"*.

ki
16 byte authentication/encryption KI of the inserted SIM card, in hexadecimal notation (32 characters) like *"00112233445566778899aabbccddeeff"*.

auth_algo
Authentication algorithm to be used with the SIM card. One of:

- *none*
- *xor*
- *comp128v1*

ciphers
List of ciphers that this modem supports, used to match requirements in suites or scenarios. Any combination of:

- *a5_0*
- *a5_1*
- *a5_2*
- *a5_3*
- *a5_4*
- *a5_5*
- *a5_6*
- *a5_7*

features
List of features that this modem supports, used to match requirements in suites or scenarios. Any combination of:

- *sms*
- *gprs*
- *voice*
- *ussd*

Side note: at first sight it might make sense to the reader to rather structure e.g. the *ip_address* or *arfcn* configuration as *"arfcn: GSM-1800: [512, 514, ...]"*,
but the more verbose format is chosen to stay consistent with the general structure of resource configurations, which the resource allocation algorithm uses to resolve required resources according to their traits. These configurations look cumbersome because they exhibit only one trait / a trait that is repeated numerous times. No special notation for these cases is available (yet).

5.5 *default-suites.conf* (optional)

The *default-suites.conf* file contains a list of *suite:scenario+scenario+...* combination strings as defined by the *osmo-gsm-tester.py -s* commandline option. If invoking the *osmo-gsm-tester.py* without any suite definitions, the *-s* arguments are taken from this file instead. Each of these suite + scenario combinations is run in sequence.

A suite name must match the name of a directory in the *suites_dir* as defined by *paths.conf*.

A scenario name must match the name of a configuration file in the *scenarios_dir* as defined by *paths.conf* (optionally without the *.conf* suffix).

For *paths.conf*, see Section 5.3.

Example of a *default-suites.conf* file:

```
- sms:sysmo
- voice:sysmo+tch_f
- voice:sysmo+tch_h
- voice:sysmo+dyn_ts
- sms:trx
- voice:trx+tch_f
- voice:trx+tch_h
- voice:trx+dyn_ts
```


5.6 *defaults.conf* (optional)

Each binary run by osmo-gsm-tester, e.g. *osmo-nitb* or *osmo-bts-sysmo*, typically has a configuration file template that is populated with values for a trial run.

Some of these values are provided by the *resources.conf* from the allocated resource(s), but not all values can be populated this way: some osmo-nitb configuration values like the network name, encryption algorithm or timeslot channel combinations are in fact not resources (only the nitb's interface address is). These additional settings may be provided by the scenario configurations, but in case the provided scenarios leave some values unset, they are taken from this *defaults.conf*. (A *scenario.conf* or a *resources.conf* providing a similar setting always has precedence over the values given in a *defaults.conf*).

Example of a *defaults.conf*:

```
nitb:
  net:
    mcc: 901
    mnc: 70
    short_name: osmo-gsm-tester-nitb
    long_name: osmo-gsm-tester-nitb
    auth_policy: closed
    encryption: a5_0

bsc:
  net:
    mcc: 901
    mnc: 70
    short_name: osmo-gsm-tester-msc
    long_name: osmo-gsm-tester-msc
    auth_policy: closed
    encryption: a5_0
    authentication: optional

msc:
  net:
    mcc: 901
    mnc: 70
    short_name: osmo-gsm-tester-msc
    long_name: osmo-gsm-tester-msc
    auth_policy: closed
    encryption: a5_0
    authentication: optional

bsc_bts:
  location_area_code: 23
  base_station_id_code: 63
  stream_id: 255
  osmobsc_bts_type: sysmobts
  trx_list:
  - nominal_power: 23
    max_power_red: 0
    arfcn: 868
    timeslot_list:
    - phys_chan_config: CCCH+SDCCH4
    - phys_chan_config: SDCCH8
    - phys_chan_config: TCH/F_TCH/H_PDCH
    - phys_chan_config: TCH/F_TCH/H_PDCH
    - phys_chan_config: TCH/F_TCH/H_PDCH
    - phys_chan_config: TCH/F_TCH/H_PDCH
    - phys_chan_config: TCH/F_TCH/H_PDCH
    - phys_chan_config: TCH/F_TCH/H_PDCH
```


6 Trial: Binaries to be Tested

A trial is a set of pre-built binaries to be tested. They are typically built by jenkins using the build scripts found in osmo-gsm-tester's source in the *contrib/* dir, see Section 3.2.4.

A trial comes in the form of a directory containing a number of **.tgz* tar archives as well as a *checksums.md5* file to verify the tar archives' integrity.

When the osmo-gsm-tester is invoked to run on such a trial directory, it will create a sub directory named *inst* and unpack the tar archives into it.

For each test run on this trial, a new subdirectory in the trial dir is created, named in the form of *run.<timestamp>*. A symbolic link *last-run* will point at the most recently created run dir. This run dir will accumulate:

- the rendered configuration files used to run the binaries
- stdout and stderr outputs of the binaries
- pcap files for processes doing relevant network communication
- a test log
- jenkins parsable XML (Junit) reports

The script in *contrib/jenkins-run.sh* takes care of related tasks such as

- creating the dir structure,
- generating md5 sums for the various tar.gz containing software builds to be tested,
- cleaning up after the build,
- saving extra logs such as journalctl output from ofonod,
- generating a final .tar.gz file with all the logs and reports.

7 Test API

TODO (in the meantime, look at src/osmo_gsm_tester/test.py, as well as suite.py, which calls the test's setup() function to get an idea)

8 Debugging

TODO: describe how to invoke *ipdb3* and step into a suite's test script

