Osmocom Testing Initiative

Harald Welte <laforge@gnumonks.org>

split NITB aftermath (the good parts)

- biggest architectural change since we started in 2008
- lots of good reasons and design improvements
  - finite state machines with proper timeouts / clean-up
  - proper 3GPP AoIP with interoperability tesing
  - no synchronous HLR database access
  - HLR access from OsmoMSC and OsmoSGSN
  - 2G/3G authentication over GERAN and UTRAN

split NITB aftermath (the bad parts)

- never-ending list of breakage
  - actual regressions of things that used to work before
  - things that were *known omissions* during the restructuring
- some commercial users stuck with SCCPlite and thus old @osmo-bsc-sccplite@
  - almost none of the new features or bug fixes there
  - no automatic testing
  - back-ports time-consuming

split NITB aftermath (lessons learned)

- overall complexity of Osomcoom cellular is quite stunning now
- absence of proper functional testing has caused massive fall-out
- the split architecture allows for betteer testing of smaller parts of the system
- my personal main focus of the last 5+ months:
  - testing, testing, testing, testing
  - testing, testing, testing, testing
  - some more testing
  - even more testing

Osmocom CNI testing (1/2)

- unit test (autotest, like we always had)
  - test individual functions / APIs of libraries / programs
  - executed during "make check" and hence before any patch can get merged
- automatized functional tests in TTCN-3
  - test *external* visible behavior on interfaces such as Abis, A, GSUP, GTP, MNCC, PCUIF, CTRL, VTY, ...
  - executed nightly by Jenkins (could be more frequently)

Osmocom CNI testing (2/2)

- osmo-gsm-tester
  - tests entire Osmoocom network with BTS/BSC/MSC/HLR/PCU/SGSN/GGSN/...
  - uses real BTS + MS hardware (over coaxial cable)
  - automatic execution multiple times per day
- interop tests
  - against NG40 RAN + CN simulator from NG4% (A / Gb / Iu level)
  - not fully automatized yet

Osmocom TTCN-3 Test Suites

- developed in 2017+2018
- compiled using Eclipse TITAN
  - uses just a command-line compiler + Makefiles
  - no IDE needed at all, don’t let *Eclipse* fool you
- containerized in Docker
- executed by Jenkins CI

Terminology

ATS	Abstract Test Suite
MTC	Main Test Component
PTC	Parallel Test Component
IUT	Implementation Under Test

Test Suite Philosophy

- test one network element (our IUT)
- test external behavior (3GPP and non-3GPP)
- emulate entire environment from TTCN-3
- don’t reuse Osmocom C-code protocol implementations in the tests
- test against independent TTCN-3 implementations!

What to test?

- successful cases
- erroneous cases (no answer, NACK, ...)
  - many difficult to reproduce with real phones/devices
- load / resource exhaustion
- spec compliance
- focus on functionality actually relevant to IUT

Why TTCN-3 + TITAN

- TTCN-3 specifically designed for telecom protocol testing
- TITAN team released many telecom protocols in TTCN-3, such as
  - BSSAP, L3 (RR/MM/CC), SMS (CP/RP/TP), SS, M3UA, SCCP, GTP, NS, BSSGP, …
  - shortens our test development cycle
  - permits us to test against known working industry implementations

Test suites for Osmocom CNI components

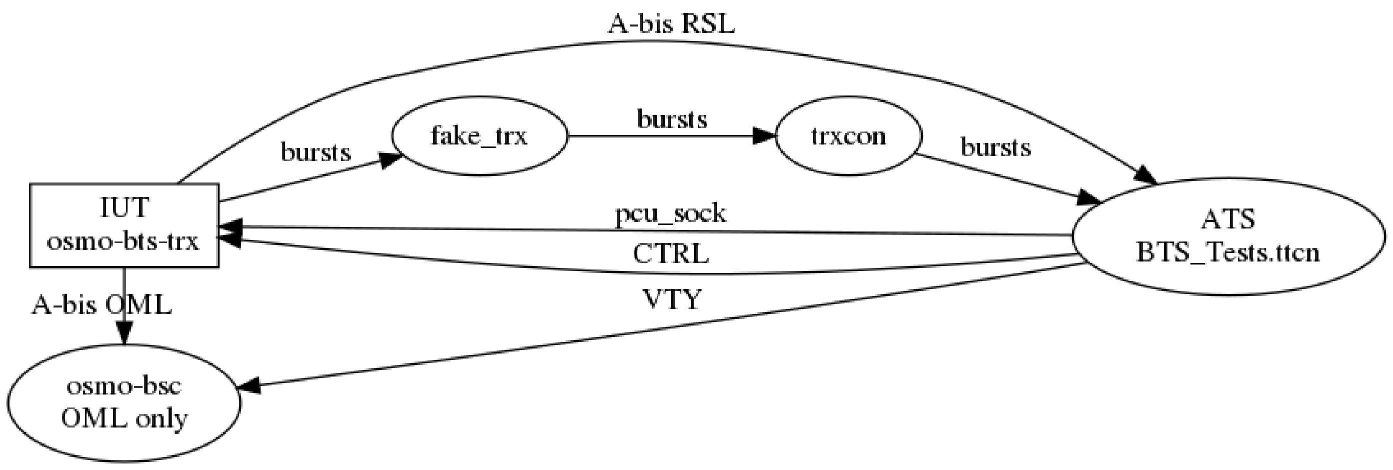
- `osmo-bts`
- `osmo-bsc` (for both AoIP and SCCPlite)
- `osmo-msc`
- `osmo-mgw`
- `osmo-hlr`
- `osmo-sip-connector`
- `osmo-sgsn`
- `osmo-ggsn`

Test suites in progress

- `osmo-pcu`
- `osmo-bsc_nat`
- `osmo-gbproxy`

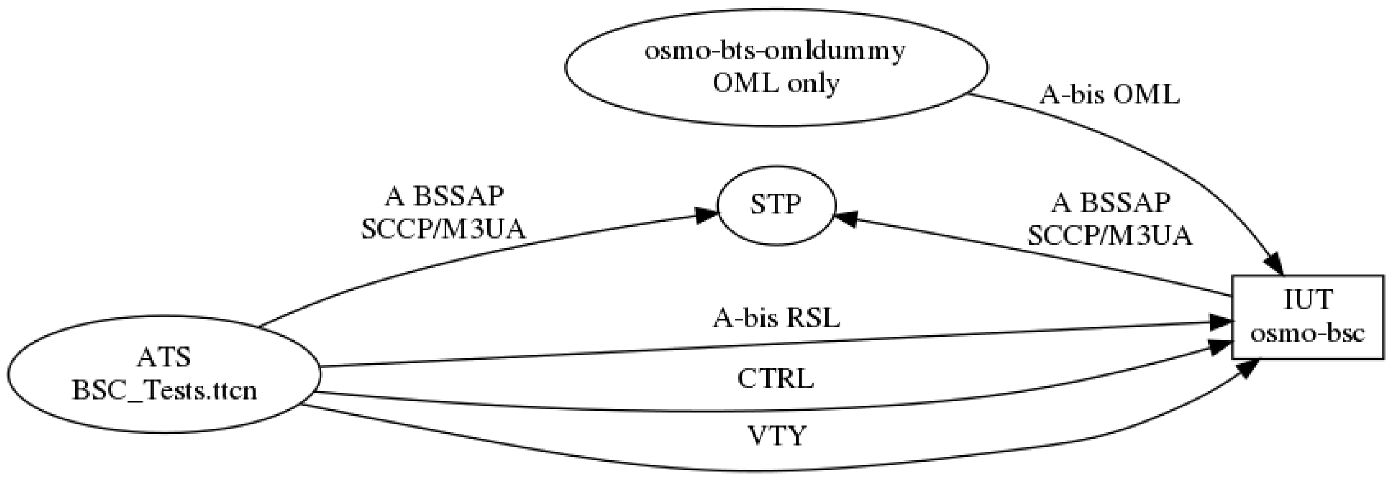
BTS_Tests.ttcn

- external interfaces
  - A-bis side: RSL (emulates BSC-side server)
  - Um side: L1CTL to control MS
  - PCU side: pcu_socket



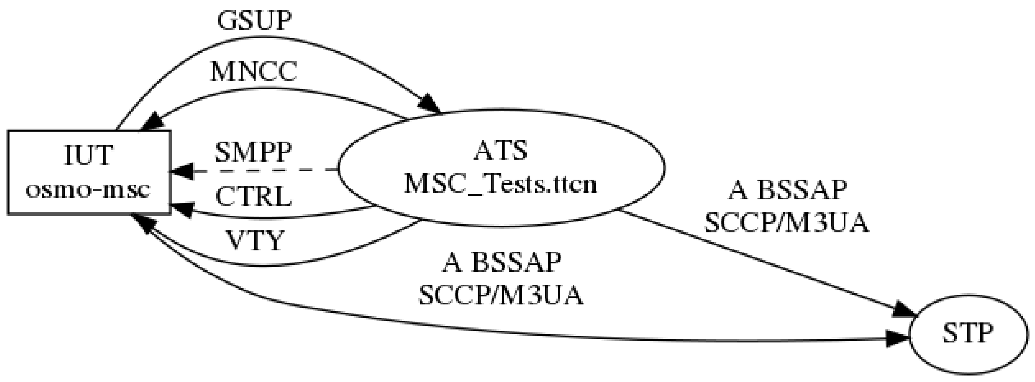
BSC_Tests.ttcn

- external interfaces
  - A-bis side: RSL (emulates BTS-side client)
  - A-side: BSSAP/SCCP/M3UA (emulates MSC-side)
  - MGW side: MGCP (emulates MGW side)



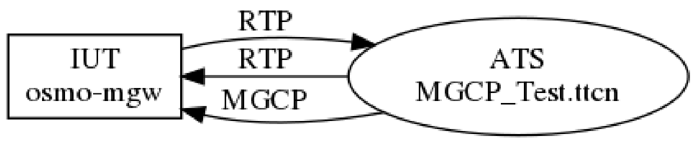
MSC_Tests.ttcn

- external interfaces
  - A: BSSAP/SCCP/M3UA (emulates BSC-side)
  - MNCC: MNCC/unix-domain (emulates ext. MNCC side)
  - MGW: MGCP (emulates MGW side)
  - GSUP (impllements HLR side)



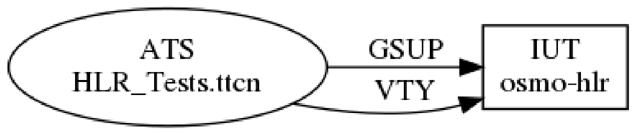
MGCP_Test.ttcn

- external interfaces
  - MGCP (emulates call agent)
  - RTP (stream source/sink)



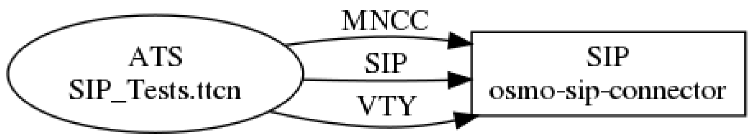
HLR_Tests.ttcn

- external interfaces
    - GSUP (emulates VLR/SGSN side)
    - VTY



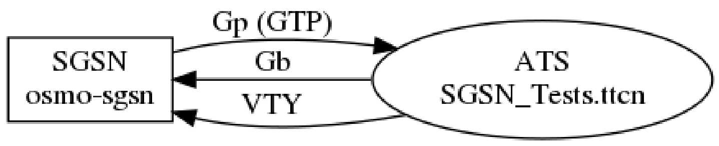
SIP_Tests.ttcn

- external interfaces
  - MNCC (emulates MSC side)
  - SIP (emulates SIP switch)
  - VTY



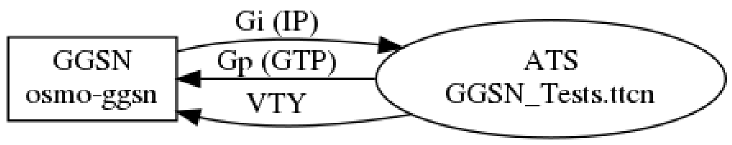
SGSN_Tests.ttcn

- external interfaces
  - Gb (emulates PCU side NS/BSSGP + MS)
  - GSUP (emulates HLR)
  - VTY



GGSN_Tests.ttcn

- external interfaces
  - Gp: GTP (emulates SGSN)
  - Gi: IP (emulates Internet)



Dockerized Setup

- one process per container
- packages either
  - IUT (e.g. `osmo-bsc`)
  - ATS (compiled docker test suite)
  - other utility (e.g. `trxcon` or `osmo-bts-omldummy`)
- why?
  - no need for local ip/network configuration
  - standardized / packaged setup on every machine
  - run older/newer versions of ATS against older/newer IUT

Jenkins CI Execution

1. update `docker-playground.git`
   a. contains `Dockerfile` for ATS + IUT
2. rebuild IUT container[s] (e.g. `osmo-bts-master`)
   a. git magic ensures re-build only if `osmo-bts.git` master changed
3. rebuild ATS container (e.g. `ttcn3-bts-test`)
   a. git magic ensures re-build only if `osmo-ttcn3-hacks.git` master changed
4. run `docker-playground/ttcn3-bts-test/jenkins.sh`
   a. creates docker network
   b. starts IUT + ATS docker containers
   c. collects test results

Jenkins CI Reporting

- junit-xml generation
- store artefacts
  - pcap file of every test case
  - ATS log file (TTCN-3 testsuite)
  - IUT log file[s] (`osmo-*.log`)
  - IUT config file[s] (`osmo-*.cfg`)
- see https://jenkins.osmocom.org/jenkins/view/TTCN3/

Further Reading

- http://git.osmocom.org/osmo-ttcn3-hacks/
- http://git.osmocom.org/docker-playground/
- http://osmocom.org/projects/cellular-infrastructure/wiki/Titan_TTCN3_Notes

EOF

End of File