ACT Manual

Wei Sun
sunweiflyus@gmail.com

Di Zhao
dzhao@cse.unl.edu

August 2, 2019

Overview

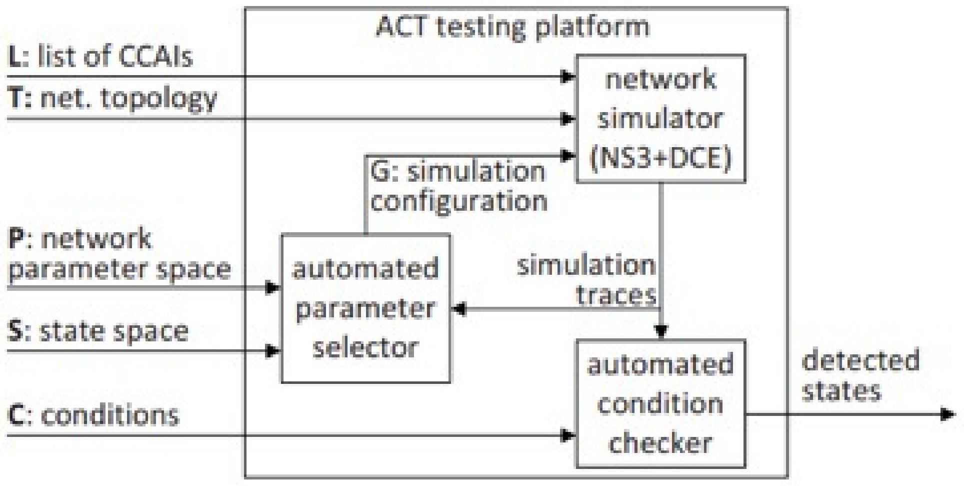
ACT testing platform
L: list of CCAIs
T: net. topology
network simulator (NS3+DCE)
G: simulation configuration
P: network parameter space
S: state space
automated parameter selector
simulation traces
automated condition checker
C: conditions
detected states

Figure 1: Architecture of a testing platform.

The implementation of ACT has four main modules: `ns3`, `DCE`, `Linux kernel`, and `ase_brain`.

For `ns3-dce`, system manipulate the packet delay in p2p channel according to given random distribution. It takes as input an input parameter configuration (i.e., input.txt).

For Linux kernel, it is a inherently user-space library used by `DCE` for running a real kernel stack. Some probing points are added here to print some target state variable in related kernel functions. Do not forget to copy a new `bin_dce (cp liblinux.so ../ns-3-dce/build/bin_dce/)` every after a new compilation of kernel source.

For `ns3-dce`, it has customized testing scripts. You could configure different typologies, algorithms in the script. More scripts example could be referred in example folder.

For `ase_brain`, it automatically generates new test input parameter configuration for verb—ns3-dce— simulation. It also processes and analyzes the log traces generate by the simulation to update related state variables coverage info. This module is the core of ACT. Adding more state variables, checking more conditions, changing new feedback algorithms, etc. are all in this module.

There already has a checker module to check required conditions or rules for the log traces generated by testing. Source files are under `ase/src/main_offline`. Take them as reference for writing different rules or conditions. It is possible to upgrade it to an online checker when processing the log traces during testing (i.e., it is easy by just do extra checking when inserting new visited states).

The whole ASE system is currently only tested well in ubuntu 14.04 but it should be OK in most ubuntu releases.

Installation

Following are all steps building the whole system, recall that the manual is build based on ubuntu 14.04. We need to install customized ns3, Linux kernel stack, DCE, three modules. Customized modules needed to be patched before installation.

Step 1. Get and install all updates and essentials

```
sudo apt-get update
sudo apt-get -y install build-essential
```

Step 2. Install required dependences for DCE-ASE

```
sudo apt-get install -y gcc g++ python  python-dev mercurial bzr cmake ...
unzip p7zip-full autoconf git cvs unrar-free libssl-dev flex bison pkg-config ...
libdb-dev libgsl0ldbl gsl-bin libgsl0-dev
```

Step 3. Build a new directory for all system files, example uses ACT

```
cd ~/
mkdir ACT
cd ACT/
git clone https://github.com/ShadowDeven/act
```

Step 4. Install customized ns3 (NS3 module)

```
cd ~/ACT
hg clone http://code.nsnam.org/ns-3.25
cd ns-3.25/
patch -p1 < ../act/patch/ns_3.25.patch
sudo ./waf configure --enable-examples -d optimized --prefix=$HOME/ACT/build ...
--includedir=$HOME/ACT/include/ns-3.25
sudo ./waf
sudo ./waf install
cd ..
```

Step 5. Install customized kernel (kernel module)

```
cd ~/ACT
git clone https://github.com/thehajime/net-next-sim.git
cd net-next-sim
git checkout sim-ns3-3.10.0-branch
patch -p1 < ../act/patch/linux_kernel.patch
```

```
make defconfig OPT=no ARCH=sim
make library OPT=no ARCH=sim
cd ..
```

Step 6. Install ns3-dce (DCE modular)

```
cd ~/ACT
hg clone http://code.nsnam.org/ns-3-dce -r dce-1.8
cd ns-3-dce
patch -p1 < ../act/patch/dce.patch
./waf configure --with-ns3=$HOME/ACT/build --enable-opt ...
--enable-kernel-stack =$HOME/ACT/net-next-sim/arch --prefix=$HOME/ACT/build
sudo ./waf
sudo ./waf install
sudo cp ../act/src/ip build/bin_dce/
sudo cp ../net-next-sim/liblinux.so build/bin_dce/
cd ..
```

By now all components finish installed. Check out your main folder, there should have four distinct folders: act, net-next-sim, ns-3.25, ns-3-dce.

Step 7. Run system for a test!
System generates binary `allinone_main` under `ACT/ns-3-dce/build` the output file `log.txt` is under the same directory. Outputs from ns3 are under `/tmp/output`.
Attention: output files will be overwritten every time!

```
cd ~/ACT/act/src/auto_perf_allinone/
make main
cd ~/ACT/ns-3-dce/build
sudo ./allinone_main
```

Data Analysis

All important data prints in log.txt file. For an analysis, one code is needed for grabbing the data you need from the log file, and another code is needed for analysis. Here gives a sample of a one test evaluation.

Figure 2 is once from a test made for evaluating the performance of a 2D simulation. All steps are listed here to make this visualization:

Step 1. Read log.txt file
For each execution of ns3 system prints the following information in the log file:

```
The number of executions 1
run the ns3 script: dce-linux
59996100
received 15Mb files
```

The number of the third line marks the time in simulation. When there is a new detected pair a line like below prints.

```
Find uncovered pairs: cwnd: 0 ssth: 6 switching_time: 0
```

If this execution is during mutation or cross-over some extra info like below also prints.

```
Output_type:ssth
Mapping empty_set:cwnd: 736 ssth: 393 switching_time: 0
[CAN] Find low limit, candidate points: 2 and 0 at granularity:512
```

Lines like the following prints the current coverage condition:

```
Every 100 times:5609 ,prev_coverage_size:5600, growth num count: 9 , total files:5000
[Coverage] map_vec.size:10
I:0, Grans:1 ,covered size:5609 ,total:1048576, coverage:0.00534916
I:1, Grans:2 ,covered size:1811 ,total:262144, coverage:0.00690842
I:2, Grans:4 ,covered size:608 ,total:65536, coverage:0.00927734
I:3, Grans:8 ,covered size:204 ,total:16384, coverage:0.0124512
I:4, Grans:16 ,covered size:68 ,total:4096, coverage:0.0166016
I:5, Grans:32 ,covered size:23 ,total:1024, coverage:0.0224609
I:6, Grans:64 ,covered size:8 ,total:256, coverage:0.03125
I:7, Grans:128 ,covered size:4 ,total:64, coverage:0.0625
I:8, Grans:256 ,covered size:1 ,total:16, coverage:0.0625
I:9, Grans:512 ,covered size:1 ,total:4, coverage:0.25
```

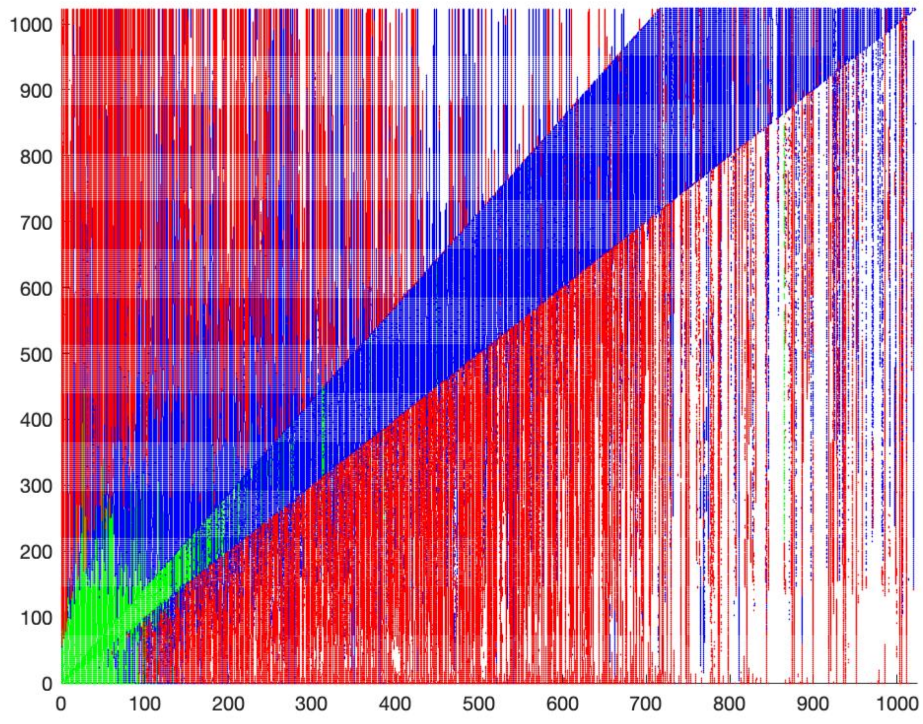

Figure 2: Coverage map for ssth and cwnd.

Grab the data needed. The code for this example can be checked out here: https://github.com/ShadowDeven/Grabber-Sample

Step 2. Do analysis
This example uses MATLAB.
The way of making a scatter: https://www.mathworks.com/help/matlab/ref/scatter.

html
The way of putting multi-plots together: https://www.mathworks.com/help/matlab/ref/hold.html

After import all grabbed data into MATLAB, following codes are used:

```
scatter(random(:,2),random(:,1),0.5,'filled','g')
hold on
scatter(mutation(:,2),mutation(:,1),0.5,'filled','b')
hold on
scatter(crossover(:,2),crossover(:,1),0.5,'filled','r')
hold off
```

Do not close pop-up during the hold.

Modify code

Changing viable for different tests: Currently, the whole system does not have a good GUI, in most cases we need to change source file and re-compile for a configuration change.

1. **State variable extension**
   Share.h file has core data structures and range definitions. Now system support following state variables: `cwnd`, `ssth`, `rtt`, `rttvar`, `ca_state`, `target`, `prev_ca_state`.For example, `#define CWND_RANGE 1024`, and loss rate `#define LOSS_UP 10`.
   To extend state variables, change this header file firstly and related functions already including the all state variables. The granularity.cc is used to process log traces generated by simulation to update current coverage. Change this file to support new add variables. You also could add assertions in function `insert_state()` when processing each new visited output state. For example, `if (cwnd > 100) print("cwnd condition volition!");`

2. **Coverage saturation**
   ASE terminates until coverage saturated. To terminate a test earlier, use command `sudo killall allinone_main`.

3. **State exchange**
   You could change the coverage saturation limit for random, feedback1, and feedback2. For example, `#define COVG_LIMIT_RANDOM 30`, where 30 is about 1.5 percent of 2048 total size (given 128 size granularity).

4. **Coverage print**
   Here is a sample of strategy, you could figure out your own modification.

```
if (TOTAL_EXECUTION > 1495 && TOTAL_EXECUTION < 1505) return 1 ;
if (TOTAL_EXECUTION > 19995 && TOTAL_EXECUTION < 20005) return 1 ;
if (inc_per < GROWTH_SSH && TOTAL_EXECUTION>20005) {
     if(re_counter < TOLERANCE) re_counter++;
     if(re_counter == TOLERANCE){
          re_counter = 0;
          return 1;
     }
     }else{
          re_counter = 0;
     }
}
```

5. **NS3 Script**
   Script dce-linux.cc in ns-3-dce/example is used to run a simple p2p link TCP 15 MB testing scenario. You could change the CA algorithm in this script. If you want to change the file size, you also need to change something in NS3 module (refer to ns3 patch file).

6. **Coverage map serialization**
   If you want to try some long-time experiments, you could leverage boost lib to do incrementally testing (i.e. keep old results and then continue later).
   Below is a tried attempt:

```
#include <boost/serialization/map.hpp>
#include <boost/archive/text_iarchive.hpp>
#include <boost/archive/text_oarchive.hpp>
#include <boost/serialization/vector.hpp>
std::ofstream ofs("/tmp/output_cubeMap_vec.txt");
boost::archive::text_oarchive oarch(ofs);
oarch << covg_map_vec;
fs.close();
ifs.open("/tmp/new_app_cubic1_correlation.txt");
boost::archive::text_iarchive iarch3(ifs);
iarch3 >> input_output_relation;
ifs.close();
```

   Basically, you could dump current coverage map into txt file or restore a coverage map from an txt file later.