ccnSim HOWTO

Giuseppe Rossini, Dario Rossi, Raffaele Chiocchetti
ccnsim@lincs.fr

March 26, 2012

1 Introduction

- ccnSim is a scalable chunk-level simulator of Content Centric Networks (CCN)[3], that we developed in the context of the ANR Project Connect.
- ccnSim is written in C++ under the Omnet++ framework, and allows to assess CCN performance in scenarios with large orders of magnitude for CCN content stores and Internet catalog sizes
- On a PC equipped with 24GB of RAM, ccnSim is able to simulate content stores up to 10^6 chunks and catalog sizes up to 10^8 files in a reasonable time.
- Is out of the scope of this document to provide a thorough description of CCN, Omnetpp or ccnSim – rather, its aim is to provide simple quick start instructions.
- We hope that you enjoy ccnSim, in which case we ask you to please cite it as [1].

2 Compiling ccnSim

We assume that you have downloaded and installed Omnetpp (version 4.1) in your home directory. In order to run ccnSim, it is first necessary to patch Omnetpp (in order to use the multi-path extension described in [1]), as follows:

```
ccnSim@sushi:~$ unzip ccnSim-0.1.zip
ccnSim@sushi:~$ cp ccnSim-0.1/patch/ctopology.h  omnet4.1/include
ccnSim@sushi:~$ cp ccnSim-0.1/patch/ctopology.cc omnet4.1/src/sim
ccnSim@sushi:~$ cd ~/omnet4.1
ccnSim@sushi:~/omnet4.1$ make
ccnSim@sushi:~/omnet4.1$ cd ~/ccnSim-0.1
ccnSim@sushi:~/ccnSim-0.1$ opp_makemake -f --deep -X patch
ccnSim@sushi:~/ccnSim-0.1$ make
```

- Note 1: the version 0.1 of ccnSim requires a 64bit kernel. We have successfully built and run ccnSim-0.1 with Ubuntu versions as old as 8.04 (provided that the parallelism is architecture 64bits) and use a Linux 2.6.32 kernel on a Ubuntu 10.04LTS for our simulations. We have not tested 32bits architectures, nor will support them (see Note 2).
- Note 2: future versions of ccnSim (currently under testing) will remove the 64bits limitation by making use of the Boost libraries.

3 Hello, (CCN) world!

Once you compiled the simulator, you're able to run your first CCN simulation. The syntax to invoke the simulator execution is the following:

```
ccnSim@sushi:~/ccnSim-0.1$ ./ccnSim-0.1  -u Cmdenv  youtube.ini
```

To start, we provide a `youtube.ini` settings file, that specify a VoD catalog as in [1, 2, 4]. `out0.node_local`

- Note 1: In the above syntax, you can omit the configuration file name in case you store your settings in `omnetpp.ini` rather than `youtube.ini`.
- Note 2: The `youtube.ini` file contains comments about the variables involved, to which we refer the reader for more detailed comments.
- Note 3: For indication, running a 1800 seconds of simulated time of YouTube catalog on a Geant network (i.e., the default scenario in `youtube.ini`) takes about 2400 seconds of real time on an Intel Xeon E5620 @ 2.40GHz equipped with 12MB cache and 24GB ram.

4 Simulation input

In omnetpp the scenario is described through "ini" files. We report here the example `youtube.ini` shipped along with the simulator sources, containing detailed and hopefully useful comments to modify the scenario description.

```
#==============================================
#YouTube-like catalog configuration for ccnSim
#----------------------------------------------

# Topologies of CCN network.
[General]
network = topologies.geant_network

#==============================================
# output
#----------------------------------------------
```

```
# By default, output in results/
output-vector-file =  ${resultdir}/out${repetition}.vec
output-scalar-file =  ${resultdir}/out${repetition}.sca

# number of runs
repeat = 1
seed-set = ${repetition}

#==============================================
# simulation duration
#----------------------------------------------
# There are three simulation states:
# 1) filling caches
# 2) transient
# 3) steady-state
#
# 1) you can start with empty caches (warmup=false,default) or
# with caches already filled with randomized content (warmup=true).
# Starting with filled caches is useful only when Zipf exponent
# s very large (say >1.5) so that caches may not even fill up
# due to very skewed requests.
#
**.warmup = false

# 2) parameters tuning intervals of confidence estimation for hit
# rate of individual caches, affecting convergence speed;
#
# We implement two convergence methods (avg|wc):
# -Average (avg): we require the convergence of the network hit rate
# -Worst Case(wc): we require the convergence of all individual nodes
#
**.convergence_type = "wc"

# Zipf percentile limit for stabilization node; i.e., we require the
# convergence statistics only for the bulk of the catalog (90% by default)
**.convergence_zipf_percentile = 0.9

# threshold of variance hit rate (wc=per node, avg=over all nodes)
**.convergence_threshold_wc = 0.001
**.convergence_threshold_avg = 0.1


# 3) simulated time to spend in steady state after transient ended
**.sim_time = 1800
```

```
#============================================
#  Catalog
#--------------------------------------------

# Global request rate, in Hz (over all network nodes).
**.lambda = 50

# Client requests can be geographically skweked (see [2] in HOWTO).
# When D==-1 uniform requests are considered (default).
**.D = -1

# Content popularity: Mandelbrot-Zipf distribution parameters
# Mandelbrot-Zipf exponent
**.alpha = 1
# Mandelbrot-Zipf plateau
**.q = 0

# Catalog size, in files
**.F = 10^8

# Average file size in chunks, geometrically distributed
**.avgC =  1000

# Number of replicas of original (ie, non-cached) content.
**.repository_num = 1

# Repository placement policy
# There are two policies (ugc,pop)
# -ugc = user generated content,
# Multiple replicas may exist, but there is no single repository
# storing the whole catalog. Rather, individual files are randomly
# assigned to a repository, that can be located behind any
# CCN router (i.e., in practice we have as many repositories
# as CCN nodes).
# -pop = point of presence,
# The number of repositories equal the number of replicas, and
# the original replicas of the whole catalog are stored in each
#  repository.
#
#
**.repository_policy = "pop"


#============================================
#  CCN node
#--------------------------------------------
```

```
# Cache size (in chunks)
**.S = 10^6

# Caching Replacement Policy {(lru)|rnd|fifo|two}, see [1] in HOWTO
**.cache_replacement_policy = "lru"

# Decision Policy {(always)|never|distance|lcd|fixP}, see [1] in HOWTO
**.decision_policy = "always"

# Strategy layer policy  {closest|all}, see [1] in HOWTO;
# other interest forwarding policies will be available in
# future releases of ccnSim
**.strategy_layer = "closest"

# Multipath cardinality,
# M=1 => shortest path routing
# M=2 => two disjoint paths, valid only for strategy_layer=closest, see [1] in HOWTO
**.M = 1

# Multipath performance,
# Flag determining if paths are precomputed and stored in an external file
# (under the directory "optimal")  or if they have to be calculated on demand.
# Path stored in file  are faster, but they need to be computed once for any
# new topology.
**.fromFile = true


# Discouraged parameter, will be removed from future versions of ccnSim.
# (do not modify but do not delete either).
**.c = 0
```

5 Simulation output

Once you have run the simulation as in the previous paragraph, you will find the following statistics. Output is organized in a number of files, following an `outID.extension` naming notation where `ID` is the repetition number and `extension` can be among the following:
In file `out0.sca`:

- DATA[i], INT[i]: number of data/interest messages handled by node i during the simulation;
- hl[i]: chunk-level hit rate at different distances with respect to the request originator (i.e., tied with the distance at which the chunk has been found);

- stretch: normalized distance with respect to the server storing persistent replicas, see [1, 2];
- cache_hit: average cache hit rate over all CCN nodes (see `out0.node_local` for individual hit rates);
- full_time: time to fill all caches (i.e., after which the transient starts);
- transient_time: time at which the transient ends (and the steady state starts, that lasts for the amount of time specified in `youtube.ini`);
- div_ratio: cache diversity ratio (see [1]).

In file `out0.node_local`:

- List of repositories with cardinality (i.e., number of replicas)
- For each CCN node i the correspondent final `HitRate` achieved at the end of the simulation.

In file `out0.node_stat`:

- For each file k downloaded from node i, the log contains three values, namely k, the effort η (see definition in [2]) i.

In file `out0.vec`:

- If the convergence criterion of the simulation is the average global hit rate (see `youtube.ini`), this file reports the time evolution of the average hit rate (0.1 second steps) until the stabilization.

6 Extending ccnSim

As our main effort is devoted to *extending* ccnSim, please consider that we have limited effort to spend on its *support* (e.g., manual, code comments, etc.),

Hence, you should consider ccnSim is a "PhD-grade" simulator rather than a "turnkey" simulator. This means that (at least) MSc- (or better) PhD-grade skills may be necessary to run and modify the simulator, and inspection of the code may help over the lack of documentation (blame on us).

For any questions, comments, suggestions, please use the ccnSim mailing list `ccnsim@lincs.fr` rather than our personal addresses.

References

[1] G. Rossini D. Rossi. Caching performance of content centric networks under multi-path routing (and more). Technical report, Telecom ParisTech, 2011.

[2] D. Rossi G. Rossini. A dive into the caching performance of content centric networking. Technical report, Telecom ParisTech, 2011.

[3] V. Jacobson, D.K. Smetters, J.D. Thornton, M.F. Plass, N.H. Briggs, and R.L. Braynard. Networking named content. In *CoNEXT*, pages 1–12. ACM, 2009.

[4] D. Rossi and G. Rossini. On sizing ccn content stores by exploiting topological information. In *NOMEN Worshop at IEEE Infocom'12,*, Orlando, FL, March 25-30 2012.