Group Centric Networking Software Release Notes

# Copyright Notice

\* Group Centric Networking

\*

\* Copyright (C) 2015 Massachusetts Institute of Technology

\* All rights reserved.

\*

\* Authors:

\* Patricia Deutsch <patricia.deutsch@ll.mit.edu>

\* Gregory Kuperman <gkuperman@ll.mit.edu>

\* Leonid Veytser <veytser@ll.mit.edu>

\*

\* This program is free software; you can redistribute it and/or modify

\* it under the terms of the GNU General Public License version 2 as

\* published by the Free Software Foundation;

\*

\* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"

\* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE

\* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE

\* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE

\* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR

\* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF

\* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS

\* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN

\* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)

\* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE

\* POSSIBILITY OF SUCH DAMAGE.

# Overview

GCN uses five protocol messages as shown in Table 1. Messages can be used for communication between GCN and an application and for communication over the air.

Table 1 – GCN Messages

|  |  |  |
| --- | --- | --- |
| **Message Name** | **Used for messages:** | **Description** |
| PULL | * Sent between GCN and applications | * Used by a group node that is a subscriber to notify GCN that it wants to subscribe to a group. * Used by GCN to notify application when there are subscribers. |
| UNPULL | * Sent between GCN and applications | * Used by a group node that is a subscriber to notify GCN that it wants to unsubscribe to a group. * Used by GCN to notify application when there are no subsribers |
| ADVERTISE\* | * Sent between GCN and applications * Sent over the air | * Used by a group node that is a producer to notify GCN that it is a source of data for the group. * Used by GCN to broadcast an ADVERTISE over the air |
| ACK | * Sent over the air | * Used by a group node to respond to an ADVERTISE sent by GCN |
| DATA\*\* | * Sent between GCN and applications * Sent over the air | * Used to send application data from source application to GCN, then sent over the air and then send from receiving GCN to the subscribing application. |

\*ADVERTISE was initially called ANNOUNCE. It is sometimes referred to as “announce” in the code provided with this release.

\*DATA was initially called PUSH. It is sometimes referred to as “push” in the code provided with this release.

The basic process for the use of these messages is shown in Figure 1 and is as follows:

1. Applications on subscribing group node send a **PULL** to GCN to indicate they want content
2. Application on source group node sends **ADVERTISE** to GCN to indicate is has the content
3. GCN sends **ADVERTISE** over the air. Note that the application sends a single **ADVERTISE** and GCN is responsible for sending **ADVERTISE** over the air periodically.
4. GCN on a subscribing group nodes send **ACK** over the in response to ADVERTISE
5. GCN on source node sends application a **PULL** when it receives an ACK
6. Application on source node sends **DATA** messages which are sent over the air by GCN
7. GCN on receiving node forwards **DATA** message OTA and/or sends to application and/or neither of these

For applications that operate in the simple GCN mode and do not use ADVERTISE and ACK messages, only items 1, 2, 5 and 7 in the list above are used. If an application no longer wants to subscribe to the content, it sends an UNPULL to GCN. If an application is no longer a source for a GID, it sends an ADVERTISE to GCN.

**Source Node**

**Application**

**Subscriber Node**

**1. PULL**

**2. ADVERTISE**

**3. ADVERTISE**

**4. ACK**

**5. PULL**

**6. DATA**

**7. DATA**

**GCN**

**GCN**

**Application**

Figure 1 – GCN Messages

# GCN Code Released

Untarring the provided release file creates a directory called GCN\_RELEASE with the following contents:

**GCN\_RELEASE/:**

/gcn

/ns3

README

GCN\_RELEASE/gcn:

Build

CMakeLists.txt

src

GCN\_RELEASE/gcn/src:

CMakeLists.txt

Common.cpp

Common.h

gcnClientBasic.cpp

gcnClientBasic.h

gcnClient.cpp

gcnClient.h

gcnClientManyToOne.cpp

gcnClientManyToOne.h

gcn.cpp

GCNMessage.proto

gcnService.cpp

gcnService.h

GCN\_RELEASE/ns3:

GCN\_Code\_Release\_Notes.docx

GCN\_network\_tests\_LLSW.cc

GCN\_network\_tests\_Wifi.cc

run\_sims.sh

wscript

Note that in the GCN\_RELEASE/ns3 directory there are two NS3 scenario .cc files:

GCN\_network\_tests\_Wifi.cc

GCN\_network\_tests\_LLSW.cc

These files are the same scenario except that one supports the use of the NS3 Wifi (802.11) model only while the other supports the use of the MIT Lincoln Laboratory Simple Wireless model. In order to use GCN\_network\_tests\_LLSW.cc, you must also install the MIT Lincoln Laboratory Simple Wireless model. If you plan to use this Simple Wireless model, see “Part IV: OPTIONAL Install Simple Wireless Model” below for instructions on how to install it.

# Release Instructions

This code operates under Ubuntu 14.04 using DCE1.5 and NS3.22.

The instructions below assume that following are already installed:

|  |  |
| --- | --- |
| **Library** | **Name to install with sudo apt-get install if not already installed** |
| boost | libboost-all-dev |
| gdb | gdb |
| bc | bc |
| mercurial | mercurial |
| cmake | cmake |
| Google protocol buffers | libprotobuf-dev  protobuf-compiler |
| pcap | libpcap-dev |

**WARNING: copying and pasting commands from this document may not always work. Sometimes invisible formatting characters are picked up, causing the command execution to fail for no visible reason. In such cases you will have to retype the command by hand.**

## Part I: Build GCN Code to run under NS3

1. **Set GCN build flags**

In GCN\_RELEASE/gcn/build run cmake making sure to set the NS3 build flag:

cmake .. -DNS3:String=ON

1. **Build the code**

make

## Part II: Install and Build DCE/NS3 (skip this if NS3.22 and DCE1.5 are already installed)

The GCN code model has been run with NS3.22/DCE1.5, NS3.21/DCE1.4 and NS3.20/DCE1.3

NOTE: It is assumed that DCE will be installed at ~/dce

1. **Return to the home directory**
2. **Install Bake**

hg clone http://code.nsnam.org/bake bake

1. **Set path variables**

Add the following lines to .profile:

BAKE\_HOME="$HOME/bake"

PATH="$PATH:$BAKE\_HOME"

PYTHONPATH="/usr/lib/python2.7:$BAKE\_HOME"

Execute the .profile:

. .profile

1. **Increase file descriptor limit to 2048**

To check the current value: $ ulimit –n

To change the current value: $ ulimit –n 2048

1. **Create a directory for DCE**

mkdir dce

1. **Configure, download and build DCE 1.5** (note: this automatically includes ns3.22)

cd dce

bake.py configure -e dce-ns3-1.5

bake.py download [See note below]

bake.py build

During download and build, you may run into missing packages. If so, install them using

sudo apt-get install *xxxx* (where xxxx is the name of the missing package)

Then after installing re-run the download or build command before proceeding to the next step.

**NOTE:** Some of the package names may not match what the dce install says is missing. Examples:

DCE name Actual Install Name

pybindgen-0.17.0.868 bzr

qt4 qt4-dev-tools

lib\_debug libc6-dbg

pygraphviz python-pygraphviz

pygoocanvas python-pygoocanvas

If the gccxml package is missing, the build may complain about pybindgen instead. Install gccxml and the pybindgen problem will probably go away.

To diagnose a build failure, you may need to use the –vvv flag to get more information:

bake.py build -vvv

## Part III: Modify NS3 and DCE to Support GCN Code

1. **Make the following modifications to the DCE code**

|  |  |
| --- | --- |
| **File to modify**  dce/source/ns-3-dce/ | **Changes to make** |
| model/libc-ns3.h | add the following:  Under math.h (about line 608):  NATIVE(ceilf)  Add the following after the math.h section (about line 614):  // LINK.H  NATIVE(dl\_iterate\_phdr) |
| model/libc-dce.cc | add the following (about line 100):  #include <link.h> |
| model/pipe\_fd.h | add the following (about line 87):  int m\_fdFlags; |
| model/pipe\_fd.cc | add the following cases to the PipeFd::Fcntl funtion  (about line 387):  case F\_GETFD:  return m\_fdFlags;  break;  case F\_SETFD:  m\_fdFlags = arg;  return 0;  break; |
| wscript | add the following lines at line 96:  ns3waf.check\_modules(conf, ['olsr'], mandatory = False)  ns3waf.check\_modules(conf, ['aodv'], mandatory = False) |

1. **Make the following modifications to the NS3 code**
   1. Modify dce/source/ns-3.22/src/wifi/model/wifi-net-device.cc to add the following:

At line 69:

.AddTraceSource ("DeviceTxBegin",

"Trace source indicating a packet has begun transmitting",

MakeTraceSourceAccessor (&WifiNetDevice::m\_txLogger))

At line 264 in the Send function:

m\_txLogger (packet, realTo);

1. **Install GCN ns3 scenario and script in dce**
2. In **~**/dce/source/ns-3-dce create a directory called gcn\_test
3. Create links to the NS3 files used for GCN

cd **~**/dce/source/ns-3-dce /gcn\_test

ln -s *{path to GCN\_RELEASE}*/GCN\_RELEASE/ns3/run\_sims.sh

ln -s *{path to GCN\_RELEASE}*/GCN\_RELEASE/ns3/wscript

1. Create links to the NS3 scenario files that you will be using

ln -s *{path to GCN\_RELEASE}*/GCN\_RELEASE/ns3/GCN\_network\_tests\_Wifi.cc

ln -s *{path to GCN\_RELEASE}*/GCN\_RELEASE/ns3/GCN\_network\_tests\_LLSW.cc

* *If using LLSW scenario file, the MIT Lincoln Laboratory simple wireless model must be installed. If you have not already done so, install that model before proceeding with the rest of these instructions.*

1. **Modify the dce wscript so that it builds the files in the gcn\_test directory**

The following additions to the wscript file will cause the dce build to include the newly created gcn\_test directory in the build.

1. **Add the following lines to the file** dce/source/ns-3-dce/wscript **at line 505.**

**You can copy the lines for the “myscripts” dir (lines 485-504) and edit those if you prefer.**

def add\_gcn\_test(bld):

for dir in os.listdir('gcn\_test'):

if dir.startswith('.') or dir == 'output' or dir == 'stats':

continue

if os.path.isdir(os.path.join('gcn\_test', dir)):

bld.add\_subdirs(os.path.join('gcn\_test', dir))

elif dir.endswith(".cc"):

bld.build\_a\_script('dce',

needed = bld.env['NS3\_MODULES\_FOUND'] + ['dce'],

target='bin/' + os.path.splitext(dir)[0],

source=[os.path.join('gcn\_test', dir)])

# Configure directories under gcn\_test dir

def conf\_gcn\_test(conf):

for dir in os.listdir('gcn\_test'):

if dir.startswith('.') or dir == 'output' or dir == 'stats':

continue

if os.path.isdir(os.path.join('gcn\_test', dir)):

conf.recurse(os.path.join('gcn\_test', dir))

1. **Add the following line to the file** dce/source/ns-3-dce/wscript **at line 713** immediately below the line “add\_myscripts(bld)” **:**

add\_gcn\_test(bld)

1. **Rebuild ns3 and dce**

Waf provides another way to build NS3 and dce. It allows you to build each separately and to build for optimized if desired (instructions for optimized build are shown at the end of the instructions)

$ cd /home/dce/source/ns-3.22

$ ./waf configure --prefix=$HOME/dce/build

$ ./waf

$ ./waf install

$ cd /home/dce/source/ns-3-dce

$ ./waf configure --with-ns3=$HOME/dce/build --prefix=$HOME/dce/build

$ ./waf

$ ./waf install

NOTE: If the build throws errors related to c++11, then run the following to configure the build:

CXXFLAGS="-std=c++11" ./waf configure --prefix=$HOME/dce/build --with-ns3=$HOME/dce/build

1. **To test that all is working, run the GCN shell script**

cd ~/dce/source/ns-3-dce/gcn\_test

./run\_sims.sh 1 0 0

Output will be written to the directory ~/dce/source/ns-3-dce/gcn\_test/output. Examine the file called GCN\_summary to see the simulation results. More information is below on running simulations and examining output.

## Part IV: OPTIONAL Install Simple Wireless Model

The GCN scenarios support the use of the MIT Lincoln Laboratory NS3 Simple Wireless model in addition to WIFI. To use the MIT Simple Wireless model instead of WIFI:

1. **Install the MIT Lincoln Laboratory Simple Wireless per the instructions that accompany that release**
2. **Modify the wscript file in GCN\_RELEASE/ns3 to support Simple Wireless**

def configure(conf):

ns3waf.check\_modules(conf, ['core', 'internet', 'mobility', 'olsr', 'aodv', 'wifi',

'simple-wireless'], mandatory = True))

def build(bld):

bld.build\_a\_script('dce', needed = ['core', 'internet', 'dce', 'mobility', 'olsr', 'aodv', 'wifi',

'simple-wireless'],

target='bin/GCN\_network\_tests',

cxxflags=['-std=c++0x'],

source=['GCN\_network\_tests\_Wifi.cc'] )

# bld.build\_a\_script('dce', needed = ['core', 'internet', 'dce', 'mobility', 'olsr', 'aodv',

'wifi', 'simple-wireless'],

# target='bin/GCN\_network\_tests',

# cxxflags=['-std=c++0x'],

# source=['GCN\_network\_tests\_LLSW.cc'])

### NOTE: The GCN scenarios do not work with the publicly available simple wireless model. The MIT Lincoln Laboratory version of the simple wireless model has additional features that are used in the GCN code.

## Part V: Run GCN Simulations in NS3

### Overview

There are two ways to run a GCN simulation in NS3.

1. Run one or more scenarios using the shell script (run\_sims.sh)
2. Run a single scenario by executing the NS3 scenario (GCN\_network\_tests\_Wifi.cc or GCN\_network\_tests\_LLSW) files

Most users will want to use the shell script to run scenarios because it offers many advantages over running a scenario using the .cc file directly:

* There are over 25 arguments that can be specified on the command line when running a scenario! It can get rather tedious to type in that entire command line to run a scenario. The shell script offers a much easier way to specify these values.
* The shell script post processes the results of a simulation. If running a single scenario using the .cc file the user will have to dig through the output files to determine the results.

Despite these overwhelming advantages, brief instructions are included below on how to run with the .cc files directly.

**1. To run scenarios by executing the shell script**

The easiest way to execute scenarios is by using the run\_sims.sh shell script.

This script is used to define the scenarios to run by creating an array of “scenario definitions”. These scenario definitions are used to set the values of the arguments to pass to the NS3 execution. An example is as follows:

scenario[0]="100;3;75;00;0;-1;000;0;10;100;40;0;100;0;0;10;20;00;00;WIFI;GCN;0;0;10;0;1"

The user can look at the run\_sims.sh shell script for further information on what each item is as this is well commented in the file.

When executing the shell script, the user can select the number of seeds to run for each scenario executing as well as the range of scenario numbers to run. The “scenario number” is the array index of the associated scenario definition line. The example above is for scenario 0. The shell script is run as follows:

./run\_sims.sh <optional-# seeds> <optional-start #> <optional-end scenario #>

The shell script can be used to run a scenario in GDB for debugging by uncommenting line 510 of the file.

**2. To run a single scenario by executing the NS3 .cc file**

To run a scenario using all default values for arguments:

../waf --run GCN\_network\_tests

To run a scenario passing argument values on the command line:

../waf --run “GCN\_network\_tests –nNodes=100”

To run a scenario passing argument values on the command line and setting the run number (affects the randomness like a seed would):

NS\_GLOBAL\_VALUE=”RngRun=1” ../waf --run “GCN\_network\_tests –nNodes=100”

To run a scenario in gdb:

../waf --run GCN\_network\_tests --command-template="gdb -ex 'handle SIGUSR1 nostop' --args %s - nNodes=100"

### Output

When a simulation runs, there are several results that are produced:

* NS3 output
* Shell script output

#### NS3 Output

A simulation produces directories for each node: files-*n* where n is the DCE node id (0 based) in the dce/source/ns-3-dce directory

files-0 contains first node’s file system, it also contains the output files of the dce applications launched on this node. In the /var/log directory there are some directories named with the virtual pid of corresponding DCE applications. Under these directories there are 4 files:

* cmdline contains the command line of the corresponding DCE application
* stdout contains the stdout produced by the execution of the corresponding application
* stderr contains the stderr produced by the execution of the corresponding application.
* status contains a status of the corresponding process with its start time. This file also contains the end time and exit code if applicable

If a pcap capture is included in the NS3 scenario script, the output of that is also in the dce/source/ns-3-dce directory where the script was run. The files are called GCN\_node\_N.pcap where N is the node id (0-based). To view the contents of these pcap traces, start wireshark and open the file.

#### Shell Script Output

As mentioned previously, the advantage to executing from the shell script is that it post processes the results. The shell script creates two directories for output in ~/dce/source/ns-3-dce/gcn\_test/output and stats. Results are appended to the summary file but all other files are overwritten for a particular scenario (M) and seed (N). The scenario number, M, corresponds to the array index used to define the scenario in the shell script.

|  |  |  |
| --- | --- | --- |
| **Output file** | **Location** | **Contents** |
| GCN\_summary | /output | Summary of results |
| sim\_out\_scenarioM\_runN.txt | /output | The information that is written to the screen when a single scenario is executed is stored in these files. |
| scenarioM\_runN\_members.txt | /stats | Lists the source nodes and group nodes selected for the scenario. |
| scenarioM\_runN\_positions.txt | /stats | Lists the positions of each node in the network as (x,y) pairs for each second of the simulation. If mobility is enabled, there is one line per second listing the timestamps and then these positions otherwise there is only one line for time 0. |
| scenarioM\_runN\_relays.txt | /stats | List the relay nodes for each second of the simulation in which traffic is active. |

### Miscellaneous

Another way to build NS3 and dce is to use waf. Using waf allows you to build each separately and you can specify to build then for optimized mode. There is no need to execute these commands now since NS3 and DCE are already built; they are provided below purely for future reference

**A. Build NS3 and DCE for debug excution:**

$ cd /home/dce/source/ns-3.21

$ ./waf configure --prefix=$HOME/dce/build

$ ./waf

$ ./waf install

$ cd /home/dce/source/ns-3-dce

$ ./waf configure --with-ns3=$HOME/dce/build --prefix=$HOME/dce/build

$ ./waf

$ ./waf install

NOTE: If the build throws errors related to c++11, then run the following to configure the build:

CXXFLAGS="-std=c++11" ./waf configure --prefix=$HOME/dce/build --with-ns3=$HOME/dce/build

**B. Build NS3 and DCE for optimized execution:**

$ cd /home/dce/source/ns-3.21

$ ./waf configure --prefix=$HOME/dce/build --build-profile=optimized

$ ./waf

$ ./waf install

$ cd /home/dce/source/ns-3-dce

$ ./waf configure --with-ns3=$HOME/dce/build --prefix=$HOME/dce/build --enable-opt

$ ./waf

$ ./waf install