Contributing to ns-3

Release ns-3-dev

ns-3 project

Jun 21, 2021

CONTENTS

This is the guide to *Contributing to ns-3*. Primary documentation for the ns-3 project is available in five forms:

- ns-3 Doxygen: Documentation of the public APIs of the simulator
- Tutorial
- Contributing to ns-3 (this document)
- Manual, and Model Library for the latest release and development tree
- ns-3 wiki

This document is written in reStructuredText for Sphinx and is maintained in the `doc/contributing` directory of ns-3's source code.

CHAPTER ONE

INTRODUCTION

The *ns-3* simulator is a discrete-event network simulator targeted primarily for research and educational use. The ns-3 project, started in 2006, is an open-source software project developing *ns-3*.

Like many open source projects, *ns-3* has a set of software repositories, sometimes called the "mainline." Changes to the mainline are made by or reviewed by maintainers. *ns-3* has a small core team of maintainers, additional maintainers who maintain a portion of the software, and end users. End users are encouraged to report issues and bugs, to be handled by maintainers. End users can also help by reviewing code proposals by others. Some end users who contribute high quality patches or code reviews over time may ask or be invited to become a maintainer of code within their areas of expertise.

A question often asked by newcomers is "How can I contribute to *ns-3*?" or "How do I get started?". This document summarizes the various ways and processes used to contribute to *ns-3*. Contribution by users is essential for a project maintained largely by volunteers. However, one of the most scarce resources on an open source project is the available time of maintainers, so we ask contributors to please become familiar with conventions and processes used by *ns-3* so as to smooth the contribution process.

The very first step is to become familiar with *ns-3* by reading the tutorial, running some example programs, and then reading some of the code and documentation to get a feel for things. From that point, there are a number of options to contribute:

- Contributing a small documentation correction
- Reporting or discussing a newly discovered bug
- Fixing an already reported bug by submitting a patch
- Reviewing code contributed by others
- Submitting new code for inclusion in the mainline
- Alerting users to code that is maintained or archived elsewhere
- Submitting a module for publishing in the ns-3 app store
- Becoming a maintainer of part of the simulator

The remainder of this document is organized as follows.

- Chapter 2 covers general issues surrounding code and documentation contributions, including license and copyright
- Chapter 3 describes approaches for contributing small enhancements such as a documentation or bug fix.
- Chapter 4 outlines the approach for proposing new models for the mainline.
- Chapter 5 describes how to contribute code that will be stored outside of the *ns-3* mainline
- Chapter 6 provides the coding style guidelines for *ns-3* code.

CHAPTER TWO

GENERAL

This section pertains to general topics about licensing, coding style, and working with Git features, including patch submission.

2.1 Licensing

All code submitted must conform to the project licensing framework, which is GNU GPLv2 compatibility. All new source files must contain a license statement. In general, we ask that new source files be provided with a GNU GPLv2 license, but the author may select another GNU GPLv2-compatible license if necessary. GNU GPLv3 is not accepted. Note that the Free Software Foundation maintains a list of GPLv2-compatible licenses.

If a contribution is based upon or contains copied code that itself uses GNU GPLv2, then the author should in most cases retain the GPLv2 and optionally extend the copyright and/or the author (or 'Modified by') statements.

If a contribution is borrowed from another project under different licensing, the borrowed code must have a compatible license, and the license must be copied over as well as the code. The author may add the GNU GPLv2 if desired, but in such a case, should clarify which aspects of the code (i.e., the modifications) are covered by the GPLv2 and not found in the original. The Software Freedom Law Center has published guidance on handling this case.

Note that it is incumbent upon the submitter to make sure that the licensing attribution is correct and that the code is suitable for *ns-3* inclusion. Do not include code (even snippets) from sources that have incompatible licenses.

2.1.1 Documentation Licensing

Licensing for documentation or for material found on *ns-3* websites is covered by the Creative Commons CC-BY-SA 4.0 license, and documentation submissions should be submitted under that license. Please ensure that any documentation submitted does not violate the terms of another copyright holder, and has correct attribution. In particular, copying of substantial portions of an academic journal paper, or copying or redrawing of figures from such a paper, likely requires explicit permission from the copyright holder.

2.2 Copyright

Copyright is a statement of ownership of code, while licensing describes the terms by which the owners of the code permit the reuse of the code.

The *ns-3* project does not maintain copyright of contributed code. Copyright remains with the author(s) or their employer. Because multiple people or organizations work on the *ns-3* code over time, one can think of the project and even individual files as a "mixed copyright" work.

Copyright can be claimed for originating files in *ns-3* or for making "substantial" changes to such files; the definition of substantial is open to interpretation. Copyright need not be claimed explicitly by a copyright statement; according to copyright laws in many countries, copyright rights are automatic upon publishing a work. Copyright cannot be claimed for all changes to a file; for instance, patches to fix small things or make small adjustments or improvements are not considered to be subjected to copyright protection.

Use of copyright statements in open source projects, as a means of author attribution, can be controversial, because having a long list of copyright statements on every file impairs readability. Also, since copyright is automatic, there is little formal legal requirement to add a copyright statement. The *ns-3* project has decided to follow the guidance provided by the Software Freedom Law Center in this regard: https://softwarefreedom.org/resources/2012/ManagingCopyrightInformation.html

2.2.1 Copyright on new files

When originating a new file, the originating author should place his or her copyright statement at the top in the header, preceding the copy of the license. An important exception to this is if the new file is copied from somewhere else and modified to make the new file; please do not delete the previous copyrights from the copyright file! See below for this case.

An example placement of a copyright statement can be found in the file `src/network/model/packet.h`:

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * Copyright (c) 2005,2006 INRIA
 *
 * 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;
```

2.2.2 Copyright on existing files

When providing a substantial feature (maintainers and contributors should mutually agree on this point) as a patch to an existing file, the contributor may add a copyright statement that clarifies the new portion of code that is covered by the new copyright. An example is the program `src/lte/model/lte-ue-phy.h`:

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * Copyright (c) 2010 TELEMATICS LAB, DEE - Politecnico di Bari
 * Copyright (c) 2018 Fraunhofer ESK : RLF extensions
 *
 * 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;
```

Here, Fraunhofer ESK added extensions to support radio link failure (RLF), and the copyright statement clarifies the extension (separated from the organization by a colon).

2.2.3 Copyright on external code copied from elsewhere

If a contributor borrows code from somewhere else (such as a snippet of code to implement an algorithm, or whole files altogether), it is important to keep the original copyright (and license statement) in the new file. Contributors who fail to do this may be accused of plagiarism.

Some existing examples of code copied from elsewhere are:

- `src/network/utils/error-model.h`
- `src/core/model/valgrind.h`
- `src/core/model/math.h`

If in doubt about how to handle a specific case, ask a maintainer.

2.3 Attribution

Besides copyright, authors also often seek to list attribution, or even credit for funding, in the headers. We request that contributors refrain from aggressively inserting statements of attribution into the code such as:

```
// New Hello Timer implementation contributed by John Doe, 2014
```

especially for small touches to files, because, over time, it clutters the code. Git logs are used to track who contributed what over time.

Likewise, if someone contributes a minor enhancement or a bug fix to an existing file, this is not typically justification to insert an `Authored by` or `Copyright` statement at the top of the file. If everyone who touched a file did this, we would end up with unwieldy lists of authors on many files. In general, we recommend to follow these guidelines:

- if you are authoring a new file or contributing a substantial portion of code (such as 30% or more new or changed statements), you can list yourself as co-author or add a new copyright to the file header
- if you are modifying less than the above, please refrain from adding copyright or author statements as part of your patch
- do not put your name or your organization's name anywhere in the main body of the code, for attribution purposes

An example of a substantial modification that led to extension of the authors section of the header can be found in `src/lte/model/lte-ue-phy.h`:

```
* Author: Giuseppe Piro  <g.piro@poliba.it>
* Author: Marco Miozzo <mmiozzo@cttc.es>
* Modified by:
*          Vignesh Babu <ns3-dev@esk.fraunhofer.de> (RLF extensions)
*/
```

Here, there were two original authors, and then a third added a substantial new feature (RLF extensions).

Please work with maintainers to balance the competing concerns of obtaining proper attribution and avoiding long headers.

2.4 Coding style

We ask that all contributors make their code conform to the coding standard which is outlined here: https://www.nsnam.org/developers/contributing-code/coding-style/

The project maintains a Python program called `check-style.py` found in the `utils/` directory. This is a wrapper around the `uncrustify` utility with configuration set to the conventions used by *ns-3*.

2.5 Creating a patch

Patches are preferably submitted as a GitLab.com Merge Request. Short patches can be attached to an issue report or sent to the mailing-lists, but a Merge Request is the best way to submit.

The UNIX diff tool is the most common way of producing a patch: a patch is a text-based representation of the difference between two text files or two directories with text files in them. If you have two files, `original.cc`, and, `modified.cc`, you can generate a patch with the command `diff -u original.cc modified.cc`. If you wish to produce a patch between two directories, use the command `diff -uprN original modified`.

Make sure you specify to the reviewer where the original files came from and make sure that the resulting patch file does not contain unexpected content by performing a final inspection of the patch file yourself.

Patches such as this are sufficient for simple bug fixes or very simple small features.

Git can be used to create a patch of what parts differ from the last committed code; try:

```
$ git diff
```

2.6 Maintainers

Maintainers are the set of people who make decisions on code and documentation changes. Maintainers are those people who have demonstrated, over time, knowledge and good judgment as pertains to contributions to *ns-3*, and who have expressed willingness to maintain some code. *ns-3* is like other open source projects in terms of how people gradually become maintainers as their involvement with the project deepens; maintainers are not newcomers to the project.

The list of maintainers for each module is found here: https://www.nsnam.org/developers/maintainers/

Maintainers review code (bug fixes, new models) within scope of their maintenance responsibility. A maintainer of a module should "sign off" (or approve of) changes to an *ns-3* module before it is committed to the main codebase. Note that we typically do not formalize the signing off using Git's sign off feature, but instead, maintainers will indicate their approval of the merge request using GitLab.com.

Note that some modules do not have active maintainers; these types of modules typically receive less maintenance attention than those with active maintainers (and bugs may linger unattended).

The best way to become a maintainer is to start by submitting patches that fix open bugs or otherwise improve some part of the simulator. People who submit quality patches will catch the attention of the maintainers and may be asked to become one at some future date.

People who ask to upstream a new module or model so that it is part of the main *ns-3* distribution will typically be asked to maintain it going forward (or to find new maintainers for it).

CHAPTER

THREE

SUBMITTING ENHANCEMENTS

This chapter covers how to submit fixes and small patches for the existing simulator and its documentation.

3.1 Documentation updates

3.2 Bug fixes

3.3 Small patches

3.4 Examples

CHAPTER

FOUR

SUBMITTING NEW MODELS

We actively encourage submission of new features to ns-3. Independent submissions are essential for open source projects, and you will also be credited as an author of future versions of ns-3. However, please keep in mind that there is already a large burden on the ns-3 maintainers to manage the flow of incoming contributions and maintain new and existing code. The goal of this document is thus to outline how you can help to minimize this burden and thus minimize the average time-to-merge of your code. Making sure that each code submission fulfills as many items as possible in the following checklist is the best way to ensure quick merging of your code.

In brief, we can summarize the guidelines as follows:

1. Understand what a code review really is
2. Be licensed appropriately
3. Follow the ns-3 coding style and engineering guidelines
4. Write associated documentation, tests, examples
5. Prepare carefully your submission
6. Pick a submission tool

If you do not have the time to follow through the process to include your code in the main tree, please see out of tree about contributing ns-3 code that is not maintained in the main tree.

4.1 Upstreaming new models

The term `upstreaming` refers to the process whereby new code is contributed back to an upstream source (the main open source project) whereby that project takes responsibility for further enhancement and maintenance.

For instance, if one were to develop a new routing protocol, a new Wi-Fi rate control, a new device model, etc. and wanted to insert it into the main ns-3 distribution, one would follow the process outlined here: https://www.nsnam.org/developers/contributing-code/ namely:

1. Understand what a code review really is and how it will work
2. Be licensed appropriately
3. Follow the ns-3 coding style and engineering guidelines
4. Write associated documentation, tests, examples
5. Prepare carefully your submission
6. Pick a submission tool

The process can take a long time when submissions are large or when contributors skip some of these steps. Therefore, best results are found when the following guidelines are followed:

- Ask for review of small chunks of code, rather than large patches. Split large submissions into several more manageable pieces.
- Make sure that the code follows the above guidelines (coding style, documentation, tests, examples) or you may be quickly asked to fix these things before people look at it again.

CHAPTER

FIVE

SUBMITTING EXTERNALLY MAINTAINED CODE

This chapter mainly pertains to code that will be maintained in external repositories (such as a personal or university research group repository), but for which the contributor wishes to keep consistent and compatible with the *ns-3* mainline.

If the contributor does not want to maintain the external code but wants to make the community aware that it is available with no ongoing support, links to the code can be posted on the ns-3 wiki contributed code page. A typical example is the graduating student who wishes to make his or her thesis-related code available but who does not plan to touch it further. See *Unmaintained, contributed code* below.

CHAPTER

SIX

SUBMITTING TO THE APP STORE

In brief, we can summarize the guidelines as follows:

1. Understand what a code review really is
2. Be licensed appropriately
3. Follow the ns-3 coding style and engineering guidelines
4. Write associated documentation, tests, examples
5. Prepare carefully your submission
6. Pick a submission tool

If you do not have the time to follow through the process to include your code in the main tree, please see out of tree about contributing ns-3 code that is not maintained in the main tree.

6.1 Upstreaming new models

The term `upstreaming` refers to the process whereby new code is contributed back to an upstream source (the main open source project) whereby that project takes responsibility for further enhancement and maintenance.

For instance, if one were to develop a new routing protocol, a new Wi-Fi rate control, a new device model, etc. and wanted to insert it into the main ns-3 distribution, one would follow the process outlined here: https://www.nsnam.org/developers/contributing-code/ namely:

1. Understand what a code review really is and how it will work
2. Be licensed appropriately
3. Follow the ns-3 coding style and engineering guidelines
4. Write associated documentation, tests, examples
5. Prepare carefully your submission
6. Pick a submission tool

The process can take a long time when submissions are large or when contributors skip some of these steps. Therefore, best results are found when the following guidelines are followed:

- Ask for review of small chunks of code, rather than large patches. Split large submissions into several more manageable pieces.
- Make sure that the code follows the above guidelines (coding style, documentation, tests, examples) or you may be quickly asked to fix these things before people look at it again.

6.2 Externally maintained models

Some projects choose to maintain their own version of ns-3, or maintain models outside of the main tree of the code. In this case, the way to find out about these is to look at the Related Projects page on the wiki: https://www.nsnam.org/wiki/Related_Projects

If you know of externally maintained code that the project does not know about, please email `webmaster@nsnam.org` to request that it be added to the list of external projects.

6.3 Unmaintained, contributed code

Anyone who wants to provide access to code that has been developed to extend *ns-3* but that will not be further maintained may list its availability on our website. Furthermore, we can provide, in some circumstances, storage of a compressed archive on a web server if needed. This type of code contribution will not be listed in the app store, although popular extensions might be adopted by a future contributor.

We ask anyone who wishes to do this to provide at least this information on our wiki:

- Authors,
- Name and description of the extension,
- How it is distributed (as a patch or full tarball),
- Location,
- Status (how it is maintained)

The contribution will be stored on our wiki at https://www.nsnam.org/wiki/index.php/Contributed_Code here. If you need web server space for your archive, please contact one of the project maintainers.

6.4 Related projects

Some projects choose to maintain their own derivative (fork) of ns-3, or maintain models outside of the main tree of the code. Some of these are listed as Forks in the *ns-3* app store. Others are listed on the Related Projects page on the wiki: https://www.nsnam.org/wiki/Related_Projects

If you know of externally maintained *ns-3* code that the project does not list in one of the above places, please email `webmaster@nsnam.org` to request that it be added to the list of external projects.

CHAPTER

SEVEN

CODING STYLE

When writing code to be contributed to the *ns-3* open source project, we ask that you follow the coding standards, guidelines, and recommendations found below.

A lot of the syntactical rules described can be easily enforced with the help of the _utils/**check-style.py_** script which relies on a working version of [uncrustify](http://uncrustify.sourceforge.net/) so, do not hesitate to run this script over your code prior to submission.

7.1 Code layout

The code layout follows the [GNU coding standard](http://www.gnu.org/prep/standards/) layout for C and extends it to C++. Do not use tabs for indentation. Indentation spacing is 2 spaces (the default emacs C++ mode) as outlined below:

```
Foo (void)
{
  if (test)
    {
      // do stuff here
    }
  else
    {
      // do other stuff here
    }

  for (int i = 0; i &lt; 100; i++)
    {
      // do loop
    }

  while (test)
    {
      // do while
    }

  do
    {
      // do stuff
    } while ();
}
```

(to be continued…)