

The Clustering Simulation Framework: A Simple Manual

Michele Mastrogiovanni

August 3, 2007

Contents

1	Acronyms	2
2	Introduction	3
3	Installation	3
4	Clustering	3
4.1	Main concepts	3
5	Classes overview	5
5.1	Algorithm.h	5
5.2	ClusteringModule.{h,cpp}	6
6	A Brief Tutorial	7
7	The Topology Generator (tg)	8
8	How to Extend the Analyzer	9
8.1	How to extend an utility class	9
8.2	How to use the utility class	10
8.3	Add skip methods	11
9	License	13

1 Acronyms

CSL Clustering SubLayer

2 Introduction

Clustering Framework is a library that can be used for developing and comparing clustering algorithms in ns2 [1].

This framework has been used for producing the results presented in the [2]. This brief document is a guide to the use of our clustering library. We do not assume responsibility for anything that can happen to you and your computer in relation to the use of this code: Use it AYOR!

3 Installation

Here are some easy steps to install the clustering extension on the ns-2.29 distribution.

1. Download and decompress ns-allinone-2.29 package in the directory of your choice.
2. Decompress ns-allinone-2.29:

```
> tar -xvzf ns-allinone-2.29.tar.gz
```
3. In the same directory apply the patch:

```
> patch -p0 < clustering-2.29.patch
```
4. Enter the ns-allinone-2.29 and compile it:

```
> cd ns-allinone-2.29/
> ./install
```

At this point in the ns-2.29 directory there is a **clustering** directory that contains the code of all the clustering algorithms we implemented. Makefile operates some changes to some files of the ns-2.29 distribution. These changes are needed to support the implementation of the different clustering protocols.

4 Clustering

In this section we illustrate the relations between the classes of our clustering library. The aim of this section is to give an overview of the classes involved in the framework and to give indications on how you can implement your own solutions.

In the following `$NS_HOME` indicates the ns-2.29 installation directory.

4.1 Main concepts

A clustering algorithm is a layer placed between the MAC and Link layers in the *mobile node* structure of ns2. The aim of a clustering algorithm is to create an overlay over the network visibility graph of a given topology.¹

¹ A network visibility graph is the graph obtained by having a link between two nodes that can communicate bi-directionally.

The MAC layer used by our simulations scripts is IEEE 802.11 (the so called “CMU extension” that comes with the ns2 distribution). This MAC layer provides primitives to send messages either as a unicast message (from one node to another specific node) or as a broadcast message from one node to all its neighboring nodes.

Some clustering algorithms proceed in steps: an algorithm can require nodes to exchange messages to create a first rich backbone and then try to reduce the number of backbone nodes by exploiting some topological information. In this case the final result of the clustering algorithm is given by the sequential application of two distributed algorithms: the first distributed algorithm is able to find a first backbone, the second one, uses the information collected to improve the choice.

To simplify the analysis of those algorithms, a centralized C++ class (the *Separator*) is responsible to run a distributed algorithm, wait for *termination* notification from each node and run the following algorithm after the reception of the notification by every node in the network. This class is not mapped on a real entity, it is just a simulator helper that allows to coordinate different simulation steps.

Each distributed algorithm of a clustering algorithm is called Clustering SubLayer (CSL). C++ classes that represent CSLs have some facilities for information exchange between clustering layers.

The simulated algorithm proceeds as follows:

- 1 When simulation starts the first CSL of each node in the network receive a *startModule* message (there are no assumptions on the ordering of execution). The start procedure launches the clustering procedure simultaneously at each node (where the execution time of each algorithm is the same simulation time for each node).
- 2 Each CSL exchanges messages with neighbors using primitives provided by the MAC layer (distributed algorithm implementation).
- 3 When a CSL installed in a node terminates its procedure (when for example the node contacted every neighbor to exchange some useful information) it calls the method *endModule*.
- 4 All information collected by the dumper of the current clustering layer are dumped on the screen.
- 5 If the current algorithm was the last one, the simulation end and the global statistics collected are printed out, otherwise the simulator restarts the execution from step (1) by using the next CSL that defines the clustering algorithm.

Each CSL can dump its own results on standard output as well as on the global result collector (a class that reports a collection of global metrics). For example, each CSL can dump its own connectivity graph along with the nodal energy consumption expressed in terms of packets transmitted and received, or in terms of the energy consumed. It is also possible to dynamically compose clustering algorithms by mean of the juxtaposition of one or more CSL.

The *template* directory contains some Tcl files that are used in the implementation of our clustering solutions. Each layer dump follows its own preamble.

The directory **tools** contains some utility software.

- **tg**: Topology Generator: A simple program that generates connected topologies with uniformly distributed scattered nodes in a rectangular area.
- **Analyzers**: A tool to manipulate results files produced by the simulations: Based on the structure of dump files, this program helps to perform different analysis of simulation results:
 - Consumption (energy, time),
 - Communication (packets, bytes, transmitted and received),
 - Network robustness,
 - Backbone properties (connectivity, shortest paths).

The structure of this software is modular: Each file parses results and produce tables by analyzing some set of metrics.

5 Classes overview

In this section we present an overview of the core C++ classes of the framework. All the classes that are part of the core of the library are contained in the directory *ns-2.29/clustering/common*.

The file *clustering-header.h* contains the definitions of the data types used in the framework.

5.1 Algorithm.h

This header contains the virtual abstract class *Algorithm*. This is a general purpose class used for the communication between sub-layers.

Any subclass of the *Algorithm* class needs to implement the following methods:

- `virtual bool supportProtocol(string protocol) = 0;`
- `virtual void * getData(string data) = 0;`

The first method is used to inquiry a class in order to know if some kind of data can be retrieved. The second method is used in order to retrieve data.

For example the module *c-sparsifier* support a protocol identified by the string: "sparsification". This protocol allows the inquirers to request informations for the attribute identified by the string "neighbors". The result of this inquiry is a pointer to a *NodeSet* containing a set of neighbors. The aim of this protocol is to communicate to another layer a set of nodes that are considered *neighbors*. In this way for example a sublayer can export to upper layers the knowledge of the visibility graph or can discard (for some reason) some neighbors from the visibility graph based on some topological consideration.

5.2 ClusteringModule.{h,cpp}

The class defined in these files contains the implementations of what is to all the clustering algorithms. Any one of them inherits from this class. *ClusteringModule* is also a subclass of *Algorithm*. It can provide data to inquiry upper and lower sub-layers.

The following is the list of the method defined by ClusteringModule and used by every CSL.

- `void receive(Packet * p, Handler * h)`. This method must be implemented in order to receive packets destined to the current sub-layer from a peer sub-layer in a neighboring node.
- `void startModule()`. This method must be implemented to start the clustering procedure of the current sub-layer, i.e., in order to initialize data and to send the very first message to the neighboring nodes.
- `void endModule()`. This method must be called by the sub-layer when the procedure is ended. When this method is called by the current working sub-layer of every node in the topology graph, the execution goes to the next sub-layer or, if the sub-layer is the highest, it ends the simulation.
- `int command(int argc, const char * const * argv)`. This is the standard procedure for receiving messages from Tcl. Remember to call the one of the base class at the end of the implementation.
- `NodeAddress getClusterHead(NodeAddress node)`: This method returns the clusterhead of the node passed as parameter. Each node can maintain a list of clusterheads for any of its neighbors (itself included).
- `setClusterHead(NodeAddress node, NodeAddress CH)`. This method sets the clusterhead for the chosen node.
- `void sendDown(Packet * p, size_t size, NodeAddress to, double maxDelay)`. Sends a packet down using the current MAC layer. If field *to* is `-1` then the packet is sent as a broadcast message. The field *maxDelay* is used to set a jitter before the transmission of the packet: If this value is not 0 then the packet is sent after a uniform random period between 0 and *maxDelay*.
- `bool isDataPacket(Packet * p)`. This method must be implemented in order to assess if a packet was intended to be received from a given CSL.
- `void prepareSendDataDown(Packet * p)`. The sub-layer must implement this method in order to put the right marker over a packet destined to a peer sub-layer of a neighboring node. The marked packet will pass through the lower layer without being modified.

It is also possible to access the following local protected variables.

- **myAddress**: The local ID of the node (the one chosen in the Tcl);
- **neighbors**: The set of the IDs of the neighboring nodes;

- **utility**: The pointer to the utility class associated to the sub-layer. The type of this member is *CommonUtility**;
- **initialEnergy, initialTime**: The energy owned by the node at the beginning of CSL and the time at which the algorithms started. These values are used to calculate the amount of energy consumed at the end of a CSL and the duration of the algorithm.

6 A Brief Tutorial

The purpose of this section is to make it easier to use the clustering framework. This short tutorial is a step-by-step guide to run and analyze the performance of one or more of the implemented clustering solutions.

A good starting point is analyzing the performance of a clustering protocol. Let us consider as an example the protocol called *WULI* [3, 2].

The Tcl scripting files used to launch the implemented algorithms are in the directory *template*.

If we want to list the features of the WULI implementation, we can do the following.

```
> $NS\_HOME/ns-2.29/ns template/WULI.tcl -h
```

This generates:

```
usage: ns WULI.tcl [options]
options (mandatory are marked with *):
* -t TopologyMainDirectory      Set Topology Main Directory...
* -N nodes                      Number of nodes of topology
* -I index                      Index for topology
  -d                            Degree version of WULI
  -k value                      Set value of k for k-rule
  -s                            Stojmenovic's version (include -d)
  -h                            This help
Default is standard WULI with k=2
```

Some options are mandatory (-t -N and -I) and some are optional. Here is an example.

```
> $NS\_HOME/ns-2.29/ns template/WULI.tcl -t topologies/ \
-N 100 -I 12 -d -k 3
```

The previous command will launch the WULI algorithm over the topology *topology/100/coordS12N100*. The algorithm will use the degree version of WULI (the weight of each node is given by the degree of the node in the visibility graph) and it will use the rule for $k = 3$ (see [2]). The output of this simulation is printed on the screen.

To get some statistically significant result we should launch a given number of simulations and write the corresponding results on a single file. This is obtained in the following way. (Here we assume that 10 is a number high enough of experiments for obtaining statistically meaningful results. The actual number—usually higher—depends on the metric investigated.)

```

> for ((i=0;i<10;i++)); do
>     $NS\_HOME/ns-2.29/ns template/WULI.tcl -t topologies/ \
-N 100 -I ${i} -d -k 3 >> result.txt
> done

```

During the simulation the list of the topologies used will be shown:

```

coordS0N100 - WULI(D,3)
coordS1N100 - WULI(D,3)
coordS2N100 - WULI(D,3)
coordS3N100 - WULI(D,3)
coordS4N100 - WULI(D,3)
coordS5N100 - WULI(D,3)
...

```

Now we have a file *result.txt* containing all the dumps of the performed simulations. If we want results for the shortest paths over the visibility graph and over the backbone as generated by the WULI algorithm we use the analyzer.

```

> tools/analyzers/bin/analyzer -a shortest-path -r result.txt \
-d topologies/100/ -o tmp/ -l 1

```

In the previous command we specify the metric to investigate, the result file, the directory where the topology file is contained (e.g., the file *coordS12N100*), the output directory and the layer to analyze (we can analyze all the layers between the first one and the one specified. WULI is composed by just 1 layer). The output of the previous command would look similar to the following.

```

Topology;Avg. SP Backbone;Avg. SP Inducted
coordS0N100;6.84687;5.851436.06869
coordS1N100;7.03556;5.651256.12919
coordS2N100;6.33172;5.038665.76545
coordS3N100;6.38283;5.281175.77702
coordS4N100;6.62242;5.374655.79038
coordS5N100;6.44222;5.249775.79596
coordS6N100;7.38121;5.701515.90828
coordS7N100;6.35394;5.105885.88846
coordS8N100;6.57596;5.586125.87006
coordS9N100;6.20424;5.240785.84097
Avg Shortest Path Graph: 5.84097
Avg Shortest Path Backbone: 6.6177
Avg Shortest Path Inducted: 5.41303

```

The output shows the current value of shortest path for each topology. At the end, the average over all single results is shown. For instance, here the average shortest path over the visibility graphs of the selected topologies is 5.84097.

7 The Topology Generator (tg)

The topology generator is a simple tool for generating connected topologies according to the generally accepted model of the unit disk graph. A given number

of nodes n are scattered uniformly and randomly in a $x \times ym^2$ rectangular area and a link is generated among two nodes if and only if their Euclidean distance is no more than the nodal transmission radius r .

The following are the parameters with which the topology generator tool can be run from the command line.

- **-n**: Specifies the number of nodes in the topologies.
- **-x**: Specifies the width of the deployment area.
- **-y**: Specifies the height of the deployment area.
- **-r**: Specifies the transmission radius r of nodes.
- **-t**: Specifies the number of topologies to be generated.

The tool will generate t topologies each of which is represented by two files: **coordSXNYYY** and **graphSXNYYY**, where X is the serial number of the topology generated and YYY is the number of nodes.

The *coord* files are files that can be included in tcl to set the position of the ns2 mobile nodes. The *graph* files are a list of pair of nodes (each one on a different line) that represent the links between nodes (i.e., the topology is represented by an adjacency list).

8 How to Extend the Analyzer

The analyzers implemented work closely together with the implemented clustering layers. They are able to recognize the dump protocol of the CSL framework and calculate some standard metrics. It is easy to extend their functionality or customize their capabilities via the general purpose statistics classes.

8.1 How to extend an utility class

When you have a clear idea of what a sub-layer (an implemented one or a new one) will dump on the output file, the first think to do is to write your custom *utility* class.

In the framework there are three main *utility* classes:

- *Utility* (`Utility.{h,cpp}`): Is the base class of every utility function. It has a *name* and the only thing it does is to write it down when the method *dump* is called.
- *CommonUtility* (`CommonUtility.{h,cpp}`): This class allows us to save information about the overhead of bytes and packets of a protocol (packets and bytes transmitted or received in unicast or in broadcast) and to save the energy and the time the protocol lasted.
- *ClusteringUtility* (`ClusteringUtility.{h,cpp}`): This class allows us to save and dump a *color* (i.e., the role) information related to each node at the end of the protocol implemented by the sub-layer. This class is useful to save informations about clusterhead or gateways. In our framework we stipulate that a clusterhead is *black*, a gateway is *gray* and a normal

node is *white*. This does not give information about how nodes are associated to each other. While the previous *utility* classes are used by the main base class *ClusteringModule*, this one must be used by the developer of the sub-layer.

- *BackboneUtility* (BackboneUtility.{h,cpp}): This class maintains information about the built backbone. For each node is possible to maintain its neighbors list in the final backbone (that should be a subset of the visibility graph neighbors list).

We suggest to use the class *CommonUtility* or *ClusteringUtility* for a sub-layer that does not output a complete backbone (partial protocols) and a *BackboneUtility* or derived for a sub-layer that produces the final backbone.

8.2 How to use the utility class

In order to use a utility class we have to declare in the Tcl file and assign it to the current sub-layer.

The easier way to perform this task is to duplicate a Tcl file in the *template* directory and customize it: each one of this files contains the operations needed to parse the parameters at the beginning and the definition of the clustering algorithm at the end.

A clustering algorithm must be defined by specifying the CSL involved:

```
begin-simulation "$pathCompleto/" "${index}" "${nodes}"

declareModule \
    {Agent/RAJARAMAN} \
    {Utility/RAJARAMAN} \
    "$rajaramanAlgo" \
    {RAJARAMAN}

declareModule \
    {Agent/MYCONNECTOR} \
    {Utility/CONNECTOR} \
    {CONNECTOR} \
    {CONNECTOR}

declareModule \
    {Agent/SHIVA} \
    {Utility/SHIVA} \
    "SHIVA([Agent/SHIVA set max-length])" \
    {SHIVA}

start-simulation
```

The only lines to be changed are those that contain *declareModule*.

The Tcl library *template/lib.tcl* allows to transparently define a stack of sub-layers as a clustering protocol. It is enough to declare the layers starting from the lower one. In the previous example we built a clustering protocol by mean of juxtaposition of three layer: A layer that calculates a dominating set

(*RAJARAMAN*), a layer that connects the clusterheads (the dominating nodes) that are at most three hops away (*MYCONNECTOR*) and finally a layer that prunes away redundant backbone edges that were created by layer 2 (*SHIVA*).

This is the more complex example of clustering protocol. Each layer gets information from the previous one and each one is started after that each node ended the protocol of the previous layer.

The parameters passed to *declareModule* are:

- The sub-layer agent (the class that implement the sub-layer protocol). See ns2 manual for further informations about Agents.
- The utility class associated to this sub-layer. This class will be accessed by the *utility* member function. Be sure to cast correctly the class before using it.
- The name of the algorithm (used in dump).
- The name of the header of the packet used by the algorithm (see ns2 manual).

You can use the layer in the sub-layer code in this way (a piece of code from *alzoubi.cpp* is shown):

```
((BackboneUtility*)utility)->setBackbone(myAddress, backboneNeighbors);
((BackboneUtility*)utility)->setColor(myAddress, myAddress == CH ? \
                                     "black" : "white");
```

8.3 Add skip methods

The analyzer has a layered structure (to mimic the behavior of the ns2 output).

If you created a new clustering sub-layer that produces some dumps, the first thing to implement is the *skip* function in the file *utility.{cpp,h}*. This implementation helps any another already implemented analyzer to skip the dump of your sub-layer to avoid problems.

Always remember to implement the main *skipLayer* function that know how to skip informations from any layer by calling other skip functions.

You can use anytime the simple class *Measure* defined and implemented in files *Measure.{cpp,h}* to perform some simple statistics calculations: Accumulate the sampled value of a metric and finally get the average or the variance. After that you can implement a simple analyzer (for some property) by mean of a new file to add to the list of analyzers. Every analyzer is contained in a file that begin with the word *analyzer*. The implemented ones are:

- *analyzer_backbone_property.{cpp,h}*: Analyze backbone properties.
- *analyzer_bytes.{cpp,h}*: Analyze bytes transmitted, received, energy, time elapsed and other metrics.
- *analyzer_clusters.{cpp,h}*: Analyze the number of cluster heads or gateway nodes (if any).
- *analyzer_degree.{cpp,h}*: Analyze the degree of nodes of a visibility graph and of the created backbone.

- *analyzer_sp*.{*cpp,h*}: Analyze shortest path in the visibility graph and in the created backbone.
- *analyzer_strongness*.{*cpp,h*}: Analyze the robustness of the calculated backbone by randomly picking a node from the backbone, deleting it and its incident edges and checking whether the resulting graph is still connected.

Each analyzer has a declaration part with for the data, followed by the initialization of the dump file to read. The input file is always closed at the end of method.

The central part of the analysis method cycles over the lines of the dump file and skip the layer information of the dump that are not related to the layer that should be analyzed. The skip utility you have previously declared will be used in this context.

In the following we show the common body of the analyzer. The first part and the last part verify if the analyzer has the right layer to analyze or if some dumping information should be skipped.

```

cout << ".";
cout.flush();

extractLayers(stack, layers);

if (selectedLayer >= (int)layers.size()) {
    for (i = 0; i < layers.size(); i++) {
        skipLayer(inFile);
    }
}
else {
    for (i = 0; i < (unsigned int)selectedLayer; i++) {
        skipLayer(inFile);
    }
    {
        ... PUT HERE YOUR ANALYZER CODE ...
    }
    for (++i; i < layers.size(); i++) {
        skipLayer(inFile);
    }
}

```

9 License

Apache License
Version 2.0, January 2004
<http://www.apache.org/licenses/>

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including

the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.
3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.
4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:
 - (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and
 - (b) You must cause any modified files to carry prominent notices stating that You changed the files; and

- (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
- (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

- 5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.
- 6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.
- 7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A

PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.
9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS,

WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

References

- [1] The VINT Project, *The ns Manual*. <http://www.isi.edu/nsnam/ns/>, 2002.
- [2] A. P. S. Basagni, M. Mastrogiovanni and C. Petrioli, “Localized protocols for ad hoc clustering and backbone formation: A performance comparison.” *IEEE Transactions on Parallel and Distributed Systems*, April 2006.
- [3] F. Dai and J. Wu, “An extended localized algorithms for connected dominating set formation in ad hoc wireless networks.” October 2004.