

# DisNet: A Distributed Framework for Graph Computation - The Manual

Ryan N. Lichtenwalter

March 25, 2011

## Contents

<b>1</b>	<b>Introduction and Conventions</b>	<b>2</b>
<b>2</b>	<b>License</b>	<b>2</b>
<b>3</b>	<b>Quick Start</b>	<b>2</b>
<b>4</b>	<b>Downloading and Installing</b>	<b>3</b>
<b>5</b>	<b>Core Components</b>	<b>3</b>
5.1	The Master . . . . .	3
5.2	Workers . . . . .	3
<b>6</b>	<b>Network Data Format</b>	<b>4</b>
<b>7</b>	<b>Software Development with DisNet</b>	<b>4</b>
7.1	Selecting a Data Type . . . . .	4
7.2	Coding the <code>process_vertex</code> Function . . . . .	5
7.3	Coding the <code>combine_data</code> Function . . . . .	6
<b>8</b>	<b>Deployment</b>	<b>6</b>
8.1	Distributing Files . . . . .	6
8.2	Starting the Master . . . . .	7
8.3	Starting Workers . . . . .	7
8.4	Parameters . . . . .	8
8.5	Checkpoints . . . . .	9
<b>9</b>	<b>Customization</b>	<b>9</b>
9.1	Adding Data Types . . . . .	9
9.2	Substituting Graph Libraries . . . . .	10
<b>10</b>	<b>Compatibility</b>	<b>10</b>

<b>11 Feature Requests</b>	<b>11</b>
<b>12 Known Bugs</b>	<b>11</b>
<b>13 Conclusion</b>	<b>11</b>

## 1 Introduction and Conventions

Welcome to DisNet and a smooth and enjoyable experience computing difficult results on large networks. In this document, you will find what is hopefully a thorough, detailed explanation of what is hopefully very simple and intuitive anyhow.

The remainder of this document will omit the full path of the location where you choose to place the DisNet directory. If you place DisNet in `/home/ryan` then naturally when we say `disnet/src/user.h` what we actually intend is `/home/ryan/disnet/src/user.h`.

## 2 License

This software is released under the GNU GPL version 3. You may copy, distribute, and modify this software as you like subject to the terms of the GNU GPL. In addition, we request that any publications for which DisNet was used to obtain results, no matter how directly, cite, *DisNet: A Framework for Distributed Network Computation* by Ryan N. Lichtenwalter and Nitesh V. Chawla.

In short, DisNet is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version. DisNet is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details. You should have received a copy of the GNU General Public License along with DisNet. If not, see .

## 3 Quick Start

This manual is designed to provide details about how to accomplish a variety of computations with DisNet, but DisNet is extremely easy to use. You download it from <http://nd.edu/~dial/software.html>, distribute `worker.sh` and optionally the grid submission helper scripts to the appropriate machines, modify `disnet/src/user.h` to select your data type and specify your algorithm, start the master with `-c` set to around 10% of the size of your network, and start workers with `-c` set to between 50 and 1500 vertices, depending on how much you trust the stability of the machines in question. Optionally, distribute your network file to individual workers or a distributed file system and specify

`-n` for however many cores you want workers to use. We think it likely that this is enough guidance to enable most people to start working with DisNet. If you desire details beyond this or are just interested in technical documentation, continue reading. Otherwise, we surmise you will get along just fine.

## 4 Downloading and Installing

You can find DisNet at <http://nd.edu/~dial/software.html>. The download includes everything you will need for compatible systems, so there is no need to worry about fetching and compiling external libraries or software packages. You can obtain and extract the archive to the location of your choice with the commands:

```
wget http://nd.edu/~dial/software/disnet/disnet.tar.gz
tar -xzf disnet.tar.gz
```

This will create a directory `disnet` containing all source files and scripts for the DisNet C++ implementation. That's it! No compiling or linking is necessary. You're ready to start working!

## 5 Core Components

The architecture consists of two fundamental components. These components are the master and the workers. The master does little work itself toward actually accomplishing the computation, but coordinates work among the workers and maintains the consistency of the results. Workers accept vertices as basic independent units of the computational task and run the same user-specified routine for each.

### 5.1 The Master

The master can run on any compatible machine with one simple requirement. Its firewall must be set to allow traffic bidirectionally through at least one port. Those of you without an IT background should not fret. In some cases, this is the default configuration for your machine. If not, or if you're uncertain, consult your system administrator for assistance. Keep in mind that the port on which the master listens is specifiable, so your options with respect to firewall configuration are quite unrestricted.

### 5.2 Workers

Workers can run on any compatible machine that can communicate via TCP. Outbound firewalls are rare, so you should expect virtually every machine where you have an account to be able to serve as a worker. In addition to machines where you have an account directly, you can engage grid resources through services such as the Sun Grid Engine (SGE), Condor, or Globus. For these

three services, you will find worker submission scripts in the `disnet/bin` directory. If you have access to computational grid resources aside from these three services, please do not hesitate to send the name of your service along with whatever details about the submission process you can gather. We would be very happy to extend DisNet by writing and distributing a submission script.

## 6 Network Data Format

The DisNet data format is designed for use with the skeleton DisNet library, which currently handles unweighted, directed networks. The format requires that the network is encoded in terms of continuous integral vertex names starting from zero. The first line of the file is the number of vertices. Successive lines describe the adjacencies of vertices starting with zero. The vertex itself does not appear as the first item on each line.

Here is an example of a 6-vertex directed graph creating a cyclic chain through vertices 0, 1, 2, 3, and 4, with vertex 5 branching of the chain at vertex 0.

```
6
1 5
2
3
4
0
```

This representation was designed to produce the smallest possible representation of the network for the purpose of fast dissemination to workers. We recognize that it is not a common format, so we provide converters for a few common formats including a simple adjacency list format and a variant of the Pajek format. These converters are located in `disnet/converters`.

If your computation requires edge weights or attributes of some sort, you can either modify DisNet to use the network library of your choice or you can modify the DisNet library to handle whatever type of network you like. Upon using an alternate library, you can use any of the network conversion or deserialization procedures it provides to employ whatever file formats it facilitates. DisNet as an architecture can handle any type of network with any type of vertex and edge attribute. The modifications are minimal so long as the library is implemented in C or C++. For details about such extensions please see Section 9.

## 7 Software Development with DisNet

The development process requires editing a single file. For the following subsections, you can restrict your attention to the file `disnet/src/user.h`.

## 7.1 Selecting a Data Type

In `user.h` you will see many different options for data types. You should select the most efficient option that will allow you to accomplish your computation. When the computation for each vertex yields a single piece of data, and the data for each vertex is of interest, you should probably select one of the `map` data types. When the computation for each vertex yields a descriptive statistic for every other vertex, and the descriptive statistics have to be combined, a `vector` is probably a better option. When the data for each vertex is not of interest, but instead some operation on that data produces a final value you seek, then a primitive scalar type may be best. To select from the many options, simply select one:

```
#define DATA_TYPE DATA_TYPE_VECTOR_INT
```

or

```
#define DATA_TYPE DATA_TYPE_DOUBLE
```

Please note that it is critically important that this line is before the inclusion of `macros.h`.

All data types including primitives will be initialized by their default constructor except for random-access data structures, which contain  $|V|$  objects each initialized by their default constructor. For example, a `double` is initialized to 0.0, a `map<unsigned int, bool>` is initialized to a map with no existing key-value pairs, and a `vector<string>` is initialized to  $|V|$  empty strings. In the rare case that this initialization behavior is undesirable, users may easily modify it by changing one line of code in `macros.h`. The process for doing so is detailed in Section 9.

## 7.2 Coding the `process_vertex` Function

The function `process_vertex` in `user.h` is constructed so that users can forget about every aspect of the framework except how to compute results for a single vertex. User code can either employ the API in the network library to achieve arbitrary unanticipated analysis, or it can call library procedures. For example, the included library already implements closeness centrality so computing closeness for all vertices requires writing only a one-line call. In the case when users want to experiment with new analytical techniques or implement new functionality, most of the coding requirements for the framework will rest in the vertex processing task.

To code this function, you should restrict your attention to the block surrounded by

```
// USER VERTEX CODE BEGINS HERE  
// USER VERTEX CODE ENDS HERE
```

In this function, you have access to the network itself via the variable `network`, the variable `data`, and the variable `vertex`. The `data` variable takes

the form of whatever data type you specified, so you can interact with it accordingly. Whatever modifications you make to that variable will be returned with the data type for eventual combination.

### 7.3 Coding the `combine_data` Function

The function `combine_data` in `user.h` is constructed so that users can forget about every aspect of the framework except how sets of results from two vertices should be combined. This snippet of code will usually be extremely minimal. It might involve merging maps, summing two vectors, concatenating two strings, or computing the maximum of two numbers. These tasks are usually straightforward, involving at most a loop expressed in three lines of code.

To code this function, you should restrict your attention to the block surrounded by

```
// USER COMBINATION CODE BEGINS HERE
// USER COMBINATION CODE ENDS HERE
```

In this function, you have access to the network itself via the variable `network`, the variable `data`, and the variable `vertexData`. The `data` variable is mutable just as in `process_vertex`, so you should store combination results in it to be returned for further computation. It contains the current state of results wherever it is being invoked, either on the master or on a worker. The `vertexData` contains the results of a computational run, a single vertex on workers and a single worker batch on the master, to be combined with the `data` variable.

## 8 Deployment

Deployment is simply a matter of copying worker submission files to the appropriate machine, starting the master, and running the worker submission scripts.

### 8.1 Distributing Files

Deploying the framework to arbitrary worker machines is slightly different depending on the type of resource or service involved. Deploying workers to local machines requires only the DisNet `disnet/bin/worker.sh` file. After placing that file on the machine, the machine is ready to serve as a worker upon invocation of `worker.sh` with appropriate parameters. To deploy DisNet with Condor or SGE resources, you will need two files on the scheduling machines. The first is `worker.sh` and the second is the appropriate submission helper script. After placing these two files on the machine, the machine is ready to submit workers to the grid.

Optionally, you can distribute a copy of the file containing the network on which you will be computing. This reduces the upfront network burden on the master. This is particularly an attractive solution for grid resources with a

shared file system. Nevertheless, distributing the file to local workers or placing it on a shared file system for grid workers manually is entirely optional.

The easiest way to copy files across machines is with the `scp` command. For local workers, from the machine containing your download, you should execute:

```
scp disnet/bin/worker.sh account@worker.remote.machine.fqdn:remotedir
```

where *account* is your user account on the remote machine and *localdir* is the remote directory into which you want to copy `worker.sh`.

## 8.2 Starting the Master

After copying all files to worker machine or grid submission machines, you can start the master. In fact, you can start the master anytime after making your modifications to `user.h`. It will persistently await worker connections. You can start the master on the master machine by running the following command from any working directory on that machine:

```
disnet/bin/master.sh port filename
```

where *port* is the port on which you want the master to listen to connections and *filename* is the file in which the network representation resides. We will discuss the parameters accepted by the master momentarily. If you have an existing checkpoint in the checkpoint directory, the master will try to load the checkpoint. If it sees that the number of vertices in the network has changed, it will report a mismatch between the checkpoint and the specified network. If you wish to begin computation on a new network distinct from the one for which you have checkpoints, you should move your checkpoints out of the checkpoint directory into a directory with a name describing the problem the checkpoints solve. Assuming the master either successfully reads an existing checkpoint, or finds now checkpoint and starts a new task, it will inform you that it is ready to accept worker connections.

```
mkdir disnet/diameter
mv disnet/checkpoint/* disnet/diameter/
```

In reality, you only actually need the last checkpoint. Prior checkpoints serve as a log of activity. You could instead run the commands:

```
mkdir disnet/diameter
mv disnet/checkpoint/\$(ls | tail -n 1) disnet/diameter
rm disnet/checkpoint/*
```

## 8.3 Starting Workers

At this point you can start you workers. In the case of local workers, you must run the command:

```
worker.sh master.ip.or.fqdn port
```

where the minimal specification is the location and port of the master. We will discuss parameters in more detail momentarily. For grid workers, you will employ batch submission helper scripts in much the same way:

```
condor_submit_workers master.ip.or.fqdn port number
sge_submit_workers master.ip.or.fqdn port number
```

The master location and port are the same as before, but these scripts require a number corresponding to the number of grid resources you want to devote to working on your problem. Naturally the number you will actually acquire are subject to the availability of those resources to you and any local policies or limitations.

## 8.4 Parameters

The master script and submission scripts take several parameters. Each set of parameters may be listed by calling the script with no arguments. Required parameters are listed after the script name, and optional parameters are listed under the script. The `v` parameter always specifies verbose output for debugging or monitoring.

```
[rlichten@mobility:~/disnet]> bin/master.sh
Usage: master.sh [options] port filename
Options: -c COMBINE  Number of vertices to combine before checkpointing (default=1)
         -v          Print verbose output for debugging or monitoring
```

The master requires as arguments the port on which it will listen for connections and the file in which the network representation resides. The only optional parameter is `c`. On the master, `c` determines how many vertex results should be computed by workers and returned before creating a checkpoint. The lower the value, the less work is lost upon failure of the master. The higher the value, the lower the disk bandwidth required. In practice, we suggest that `c` be between 5 and 10 percent of the size of the network. For a million node network, checkpoint every 50 to 100 thousand vertices.

```
[rlichten@mobility:disnet]> worker.sh
Usage: worker.sh [options] host port
Options: -f NETWORK  Filename of file containing network data
         -n THREADS  Number of worker threads (default=1)
         -c COMBINE  Number of vertices to combine before sending (default=1)
         -v          Print verbose output for debugging or monitoring
```

All workers accept these parameters, including workers submitting through grid submission helper scripts. The parameter `f` indicates the local copy of the network to load, if one is available. If this parameter is not specified, then the worker will simply request that the master send a copy. The `n` parameter is the number of threads the worker should use. On multicore local workers, this number can be equal to the number of unused cores. Many grid resources are, by default, one core per job. If you find that your system is not, you can submit multithreaded jobs by specifying the value of the `n` parameter. The `c` parameter controls the number of vertex data combinations the worker should run before sending results back to the master. On workers, the combination parameter designates the number of vertices to finish and combine before sending data to the master. A lower parameter value decreases wasted worker time when the

worker fails; higher values reduce the computational and network bandwidth demands on the master. If `c` is unspecified, every result from that worker will be sent back to the master immediately after it is computed. In practice, this parameter should be set according to your confidence in the stability of the worker. For Condor workers, lower parameter values are desirable, such as 50, 100, or 150. For local workers under your control, the value can be very high, such as 500 or 1000.

## 8.5 Checkpoints

DisNet employs checkpoints to minimize the worker productivity losses if the master fails. The master creates a checkpoint in one of three circumstances: (1) every time a specified number of vertices has been successfully processed, (2) when all vertices have been successfully processed, or (3) upon detecting Ctrl+C. When the master is restarted after failure, it searches for the latest existing checkpoint, loads the vertex status list, recovers results into the appropriate data structure, and listens for connections from new workers. Each checkpoint includes all the necessary status information to resume computation upon master failure along with the results of computation so far. Checkpoints are placed in the `disnet/checkpoint` directory with a POSIX time timestamp. Only the last checkpoint file is required to resume computation; the others may be deleted at your discretion. The final checkpoint also serves as the location where results reside. Results will be outputted in a format corresponding to the ASCII serialization routine for the data type you selected. For instance, for a map:

```
< disnet/checkpoint/123456789.checkpoint tail -n $|V|$
0 0.5
1 0.75
...
```

indicates the value 0.5 for vertex 0 and 0.75 for vertex 1.

## 9 Customization

For almost all conceivable computation, you will only have to modify code in `user.h`, but those looking to customize some aspect of the implementation for very specific purposes are welcome to do so.

### 9.1 Adding Data Types

To add a data type, you must first modify `disnet/src/user.h` to provide the new macro option for the data type. The macro should associate the new data type with the next unused integer, 13 in the case below. For instance, to add a data type corresponding to a `vector<map<vertex_t,double>>` the line might appear thus:

```
#define DATA_TYPE_VECTOR_MAP_DOUBLE 13
```

Then, to actually implement support for the data type, you must modify `macros.h` with the addition of three lines. The first line checks for user specification of your data type, the second line provides the declaration for the data type, and the third line is the constructor. For the `vector<map<vertex_t,double>>` this might look as follows:

```

1 #elif DATA_TYPE DATA_TYPE_VECTOR_MAP_DOUBLE
2 #define TYPE vector<map<vertex_t,double>>
3 #define INITIALIZE(var) var = new TYPE( network.vertexCount() )

```

You need not worry about defining destructor routines, code to send or receive your data type, or code to cast your data type. These are all defined lower in `macro.h` in a generic way. If your data type contains a C++ class that does not admit its own serialization or string conversion, you may also have to provide serialization code, however.

## 9.2 Substituting Graph Libraries

The DisNet framework itself is independent of any particular form of network or network library. Nevertheless, it is distributed with a skeleton graph library sufficient to perform many computations on unweighted, directed networks. If, in preference of extending this library, you prefer to substitute an additional library, the required effort is minimal so long as that library is implemented in C++.

In the files `master.cpp`, `worker.cpp`, and `macros.h` you must substitute all functions, including stream extraction and insertion functions, with their equivalent in the library you select. Almost all of these substitutions involve substituting the function `network.vertexCount()` with their equivalents. The remaining substitutions are equally trivial. Future versions of DisNet may make this even easier by storing the result of a single call of `network.vertexCount()`. Future versions of DisNet may also be distributed with built-in support for other libraries, such as the Boost Graph Library.

## 10 Compatibility

This implementation of DisNet was designed with speed as the principal concern and portability as a close second. This made dynamically compiled C++ wrapped with Bourne shell scripts a natural choice. So long as machines are POSIX compliant with basic GNU tools such as `tar` and `make` they can serve as workers. This may include Linux, BSD, Solaris, and even Windows machines with Cygwin. Despite this claim, we provide a list here of configurations on which we have tested this implementation of DisNet.

- Architectures
  - x86
  - x64

- Operating Systems
  - Linux
    - \* RHEL 4
    - \* RHEL 5

If you attempt to deploy a DisNet worker on a different system, please let us know what your results are, so we can make a note in this section and try to overcome whatever difficulties may arise in the next release.

## 11 Feature Requests

1. Add a flag, `-z`, to the master that directs it to compress its checkpoint files as it writes them to disk.

## 12 Known Bugs

Despite the listing below, we have performed extensive testing with various graphs of various sizes. None of the problems we encountered in any way affect the quality or consistency of results. They may simply cause ungraceful termination.

1. Upon detecting `Ctrl+C`, workers may encounter a segmentation fault. This may be due to poor interaction between multithreading and signal handling. Status: OPEN

If you suspect a bug or encounter any problems using DisNet, please report them to [rlichten@nd.edu](mailto:rlichten@nd.edu). We will add them to this list and work to fix them as quickly as possible for subsequent release.

## 13 Conclusion

We sincerely hope that this manual has been helpful. We look forward to making improvements to this implementation of DisNet based on community feedback. We also invite the development, and optionally the submission to us, of alternate implementations. If you enjoy the software and find it helpful, have a question, want to report a bug, or want to suggest a new feature, please send your emails to [rlichten@nd.edu](mailto:rlichten@nd.edu). Thanks for your interest, and happy computing!