3nuts

User Guide

February 23, 2008

Peter Mahlmann, mahlmann@upb.de
Thomas Janson, tjanson@upb.de

Contents

1 Introduction

3nuts ['frE 'n&ts] is a peer-to-peer network combining the benefits of reliable random graphs and semantic search trees. The goal of 3nuts is to overcome the restricted query languages induced by the use of distributed hash tables. In most of the currently used peer-to-peer networks these distributed hash tables do not support the efficient exploration of the semantic neighborhood of a data entry. In 3nuts, semantic relationships of data are preserved, because peers are assigned to the data and not vice versa.

3nuts will allow nontrivial lookups, like prefix search for example. All network operations in 3nuts are local and distributed, i.e. simple handshake operations maintain the network structure. Besides this, 3nuts provides fair load balancing, fast data access and guaranteed robustness, proved by rigorous analysis.

A note on the provided software

The purpose of the presented software is to build and maintain a peer-to-peer network adapting to the data provided in this network. The data itself is managed by an application which runs on top of the 3nuts network. 3nuts is designed such that the application is completly independent of the network layer. Therefore, we only provide a minimalistic *test application* which generates some virtual random data at every peer, so that there is some data defining a tree structure the peer-to-peer network can adapt to.

2 Installation

The 3nuts software is implemented using Java 5. For running 3nuts, the Java Runtime Environment in Version 1.5 is needed. For compiling the sources the ANT package[1] is required additionally. Currently, 3nuts does not rely on further external packages or libraries.

2.1 Compiling 3nuts

This section describes how to compile the 3nuts sources and gives instructions for Linux and Mac OS X operating systems as well as the Microsoft Windows operating system. Note that you can skip this Section if you obtained the Jar distribution of 3nuts.

There are two basic possibilities to compile the 3nuts sources and generate executable files. The first possibility is generate executables into the `bin/` subdirectory of the 3nuts source distribution. To do this, ensure that the current working directory is the main directory of the 3nuts source distribution. To start the compilation process simply type:

```
>make
```

If everything went fine, the output in the console should look like this (on a Unix based system):

```
ant -buildfile Ant.XML
Buildfile: Ant.XML

init:
    [mkdir] Created dir: /Users/naldo/3nuts/bin

compile:
    [javac] Compiling 265 source files to /Users/naldo/3nuts/
        bin
```

[1] see `http://ant.apache.org/`

```
     [copy] Copying 30 files to /Users/naldo/3nuts/bin

BUILD SUCCESSFUL
Total time: 8 seconds
```

The other possibilty is to create a stand-alone Jar archive. To generate the Jar archive type:

```
>make 3nuts
```

what should generate output like this:

```
ant -buildfile Ant.xml dist3nutsSim
Buildfile: Ant.xml

init:
    [mkdir] Created dir: /Users/naldo/3nuts/bin

compile:
    [javac] Compiling 265 source files to /Users/naldo/3nuts/
        bin
     [copy] Copying 30 files to /Users/naldo/3nuts/bin

dist3nutsSim:
    [mkdir] Created dir: /Users/naldo/3nuts/dist
      [jar] Building jar: /Users/naldo/3nuts/dist/3nuts
          -20061212.jar

BUILD SUCCESSFUL
Total time: 9 seconds
```

The generated executable Jar-File can be found in the `dist/` subdirectory. The filename is 3nuts attached with the date of the compilation (`3nuts-20061212.jar` in our example).

2.2 Removing 3nuts

In order to uninstall the 3nuts software simply delete the installation directory.

3 Usage

This chapter gives a brief overview of how to use 3nuts and create a peer-to-peer network. In particular, the following steps are illustrated:

- Startup of the 3nuts application (3.1)
- Creation of a 3nuts network (3.2)
- Preferences of a 3nuts network and the application (3.3)
- Visualization of a single 3nuts peer (3.4)
- Visualization of a 3nuts network (3.5)

3.1 Application Startup

We assume the user to have write access to the installation directory of 3nuts, since 3nuts will write preference files into this directory. Then, 3nuts can be started using the following command when you have compiled the sources on your own:

```
>./run_3nuts
```

This starts a small script provided with 3nuts. It will start the network trace window with default parameters for the network properties.

In case you downloaded the Jar distribution of 3nuts or you have generated a Jar archive as described in the previous chapter, you can start 3nuts with the following command[1]:

```
>java -jar 3nuts-20061212.jar
```

If the startup fails or the simulation is started without parameters (i.e. in the case you are using the Jar distribution for the first time), the preferences window (see Figure 3.1) will show up first. The following parameters can be adjusted in the preferences window:

[1] note that the filename of the jar file may be different in your case.

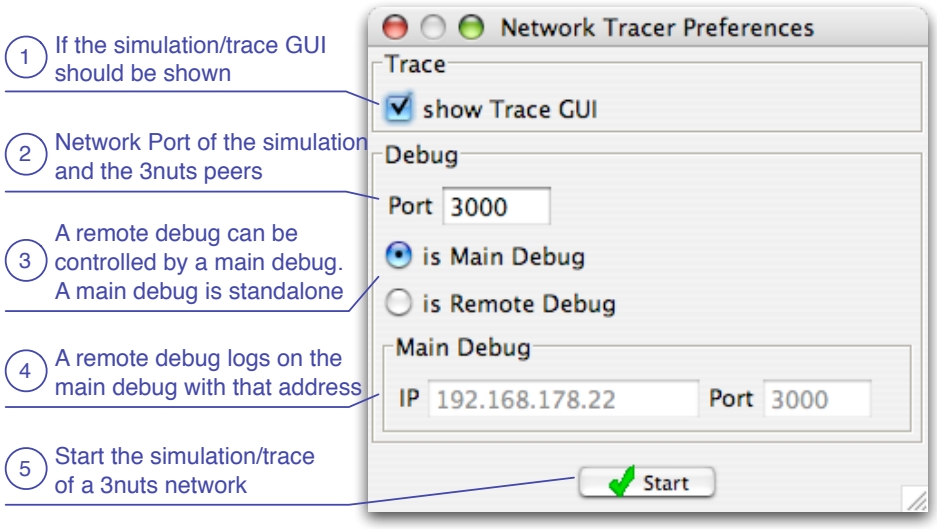


Figure 3.1: Creation of a 3nuts network simulation/trace

1. If the check box show Trace GUI is selected, the 3nuts trace window will be shown when you have left the preferences window. Otherwise, the 3nuts network will be started without graphical user interface.

2. All 3nuts peers created using the 3nuts trace window (see Section 3.2) will share a network socket. If the chosen port is occupied, the window cannot be closed using the Start button.

3. There are two different types of debugs: The main debug is a stand alone application able to trace and control a complete 3nuts network. Several remote debugs can be connected to a main debug. Then, a distributed 3nuts network can be created on several connected debugs. The complete 3nuts network can only be visualized in the main debug in this case. So, we recommend choosing is Main Debug here.

4. If a remote debug is selected in ③, the IP address and port number of the remote debug has to be entered here. The remote debug will connect to the main debug running at the specified IP and port number.

5. Use the start button ⑤ to leave the preferences window and start the 3nuts trace window. Once again note, that a free port has to be chosen before using the start button.

3.2 Creating a 3nuts Network

Once you left the preferences window using the start button, the 3nuts trace main window will appear (see Figure 3.2). In this window, you can create a 3nuts network and start all other simulation and tracing tools including 3nuts network trace, 3nuts peer trace, simulation of network dynamics, etc..

The socket address used by the simulation and the 3nuts peers is displayed in the header of the frame ①. The upper list ② shows tracers running at other hosts which are connected to this trace. Usually, there will be only one trace, i.e. the one you have just started.

The lower list ③ shows a list of all created 3nuts peers. Each entry consists of the ID and network address of a peer. Note that it is possible to run several peers on the same host using the same IP address. Therefore, each 3nuts peer has an additional *local address* which is shown in the last column, labeled Local. Even in the case 3nuts has just been started, there may be some active peers already (the number of peers created at startup can be changed in the startup script). The current number of 3nuts peers in the network is always shown next to the list of peers ⑤.

In order to create some 3nuts peers, the number of new peers has to be entered into the text box ⑥. Then, this number of 3nuts peers can be created using the Create button ⑨. If necessary, the properties of the created peers can be set in a dialog showing up when using the Properties button ⑦. Every peer is created combined with an application which uses the peer for publishing some virtual data. The application preferences can be adjusted in a dialog which can be reached using the 3nuts Appl. button ⑧. More information about the preferences can be found in Section 3.3.

In order to remove some 3nuts peers from the network, simply mark these in the list of peers ③ and click the Remove button Ⓐ.

A double click on a peer in the list of peers ③, or selecting one peer in the list and then using the Show button, a new window showing detailed information about this peer (e.g. the peers view of the 3nuts network) will appear. The contents of this window are described in Section 3.4.

For the purpose of debuging and testing, 3nuts debug tool allows to view the global structure of the whole network. This global view window shows up when pressing the button Show ④. For further information about this window see Section 3.4.

It is also possible to measure the characteristics of a running 3nuts network. The drop down menu Measurements ⓪ allows to activate the measurement tools (e.g. hop distances in the network, state of load balancing, degree distribution of the peers).

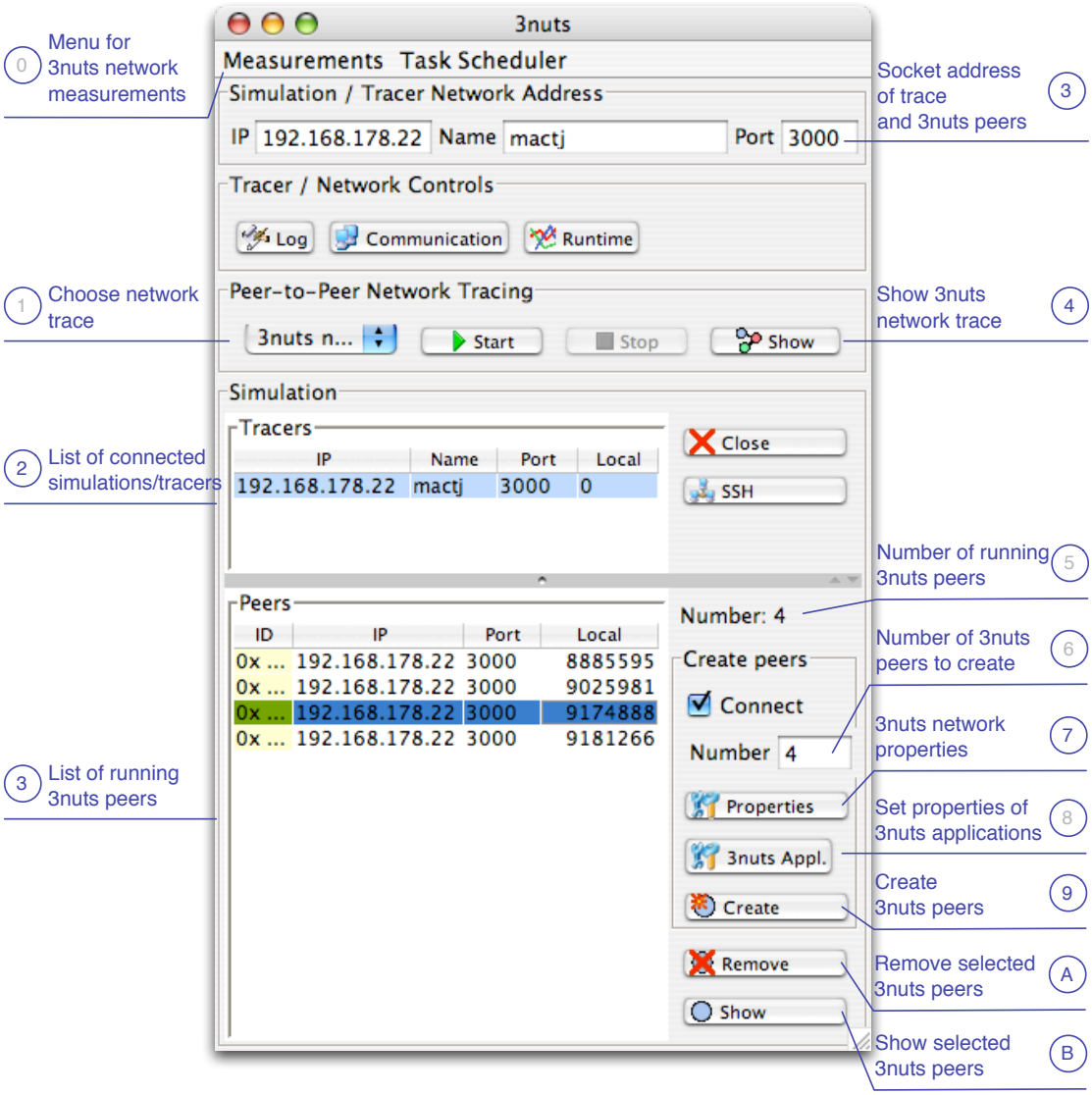


Figure 3.2: Main window allowing to create and trace a 3nuts network.

3.3 3nuts Preferences

This Section describes how to change the preferences of the 3nuts peer-to-peer network and the application running on top of it.

3.3.1 Network Preferences

Figure 3.3 shows the *Network Properties* window. The properties are structured hierarchically ①. The subgroup Random graph for example contains all properties of the random networks used to build nodes of the 3nuts network tree. Selecting an element in the left row of the Network Properties window, the corresponding parameters and their current values are shown in the right column ③ of the window. To modify one of the values, double-click on the value in the right column ②, enter the desired value and press Return.

When all properties are adjusted, press the Apply button ④ to apply the changes and close the dialog. To create a new peer-to-peer network using the modified parameters, you have to remove all peers currently in the 3nuts network. The reason for this is, that peers connecting to an existing network will take over the parameters from the peer they are using to connect to the network. This is necessary to ensure that all peers of 3nuts network use the same parameters.

3.3.2 Application Preferences

The tree structure of a 3nuts network is defined by the data of the application running on top of the network. Therefore, running a 3nuts network without an application is not reasonable. So, every 3nuts peer is created in combination with a small test application. This application publishes some random virtual data in the created 3nuts network. The behavior and virtual data can be adjusted in the dialog shown in Figure 3.4.

On top of the Application Preferences window a tree for the application data can be chosen. Open the drop down list Tree Type ⓪ to switch between different types of trees. The options for the currently selected tree type are shown in the panel below (Predefined Tree in the figure).

Predefined Tree This type of tree is a user defined tree which can be created, loaded, and edited within this 3nuts tracing tool. The predefined trees are saved in the subdirectory `predefinedTrees/`. Previously saved trees can be loaded using the Choose... button. Furthermore, they can be modified with a visual editor which will show up when clicking on the Edit... button. To allow large amounts of data in

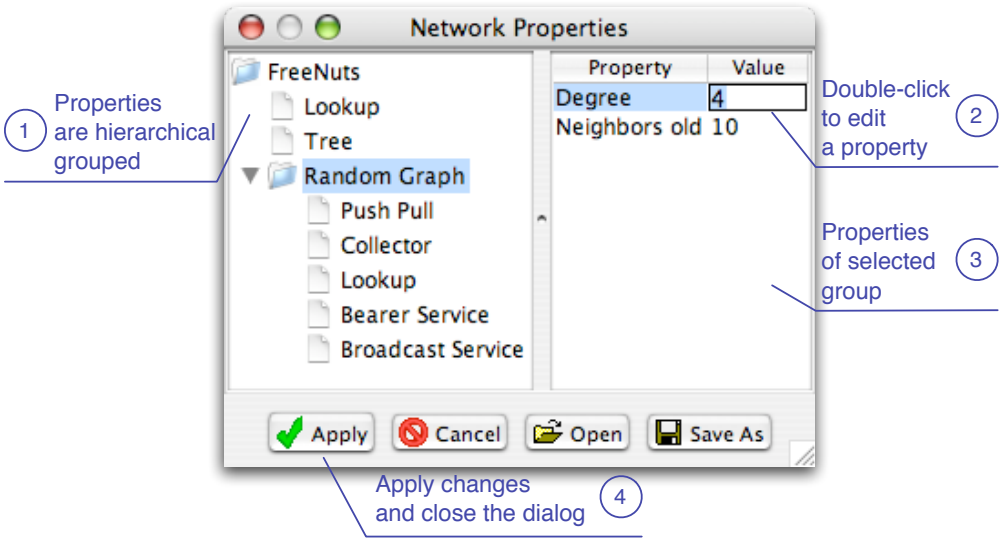


Figure 3.3: The network properties window.

a network without defining a tree by hand, every tree can be extended by *endless subtrees*, typically these are 0-1-binary trees.

The identifiers of the virtual data items created by the application and will describe a random path starting at the root of the tree. The path is generated recursively by starting in the root of the tree and then randomly choosing one of the possible subtrees. The value entered in Depth Probability ③ denotes the probability with to continue the random at the next higher level (note that 3nuts allows to store data in non-leaf nodes).

Binary Tree A balanced binary tree which will have in each node the same number of data elements in both children. The only option is the number of data elements in the tree. When the tree is created by applying and closing the dialog, each data element of the tree assigned to a 3nuts application is removed from the tree to assure an equal weighted tree in the created network. Therefore, if the product of peers and data elements stored per peer is greater than data elements in the network, some peers (to be more specific: those who were created last)will not store data elements. So, in order to create a balanced binary tree, the product of peers and data elements stored per peer should be greater equal than the number of data elements in the defined binary tree.

File List Tree This is a tree loaded from a ASCI text-file. Each line in such a file corresponds to a path in the tree. The single characters in the line are the labels of the nodes in the tree. So this is a character tree. Example trees in the directory

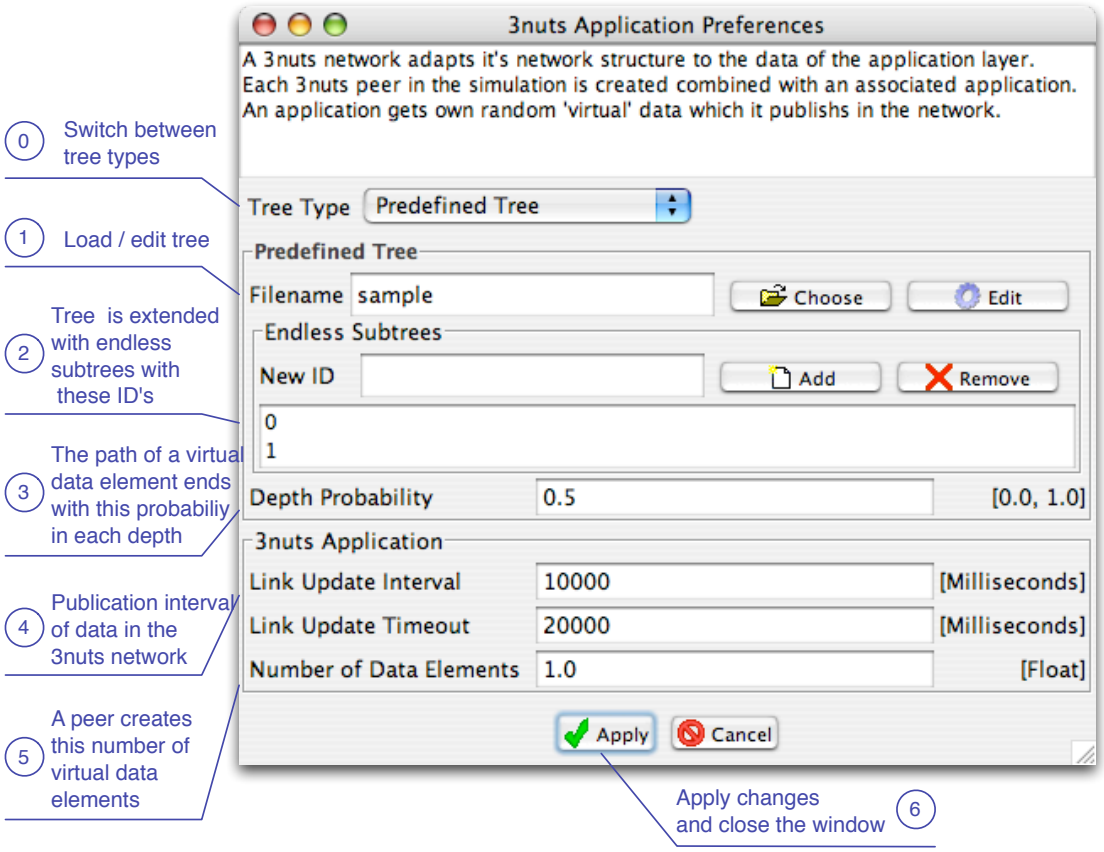


Figure 3.4: The 3nuts application preferences window.

`predefinedTrees/` are dictionaries like the popular and freely available one of *iSpell*. Once the tree is loaded, each data path used in a 3nuts application will be removed from the tree so that it can be only used once. Open and press apply in the application dialog again to reload a fresh complete tree instance.

Furthermore, the following application specific parameters can be adjusted:

Number of Data Elements The number of data elements, the application generates at a peer, can be defined in ⑤. If the value is a natural number each application generates that fixed number of data elements. It is also possible to enter a probability p. Then the number of data elements is determined by a random variable with expectation $1/p$.

Link Update Interval Link update times. A peer storing a data item, publishes this data item in regular intervals defined in ④. More precisely it connects to application of the peer responsible for the data item. If a peer responsible for a data item does not receive such an update for the time specified in Link Update Timeout, it will remove the link for the corresponding data element.

3.4 3nuts Peer

The 3nuts software allows to trace the inner state of every 3nuts peer. Figure 3.5 shows the *peer trace window* of a 3nuts peer. The top of the window shows ID information such as IP and local adress of the traced peer. The main part of the window is organized in tabs. These tabs are:

Tree Allows to view the peers local view of the tree.

Application Shows information about the application running on top of 3nuts.

Messages Shows Messages of the 3nuts network.

Neighborhood Shows the neighborhood of the traced peer.

Operations Allows to perform basic network operations (join, leave, etc.).

The informations shown in the Messages, Neighborhood, and Operations are self explanatory and therefore there is need no further description here. The contents of the Tree and Application tabs are described in the next sections.

3.4.1 The Tree tab

The Tree tab shows the peers local view of the 3nuts network. The displayed tree can be resized using the Scale slider ① and copied to the clipboard using the Copy button ③. Clicking on a node of the tree will show detailed information about that particular node on the left side of the tab ⓪. Which information is displayed depends on the color, i.e. the peers role in the selected node. The colors have the following meaning:

Blue The peer is participating in the corresponding random network, i.e. the node of the tree is part of the path the peer was assigned to by the load balancing mechanisms. Such a node will also be referred as *path node* in the following.

Green The traced peer is not participating in the corresponding random network. Instead of this, it knows a peer (a so called *branch link*) who is participating in the random network. Branch links are needed to allow efficient routing in the network.

Red The traced peer has been shanghaied to be responsible for this node of the tree, i.e. no peer has been assigned to this node by the load balancing mechanism. However, the traced peer is the one who "*feels*" most responsible.

Orange There is no random network beneath this node of the tree, i.e. that potion of the tree currently only exists locally in the traced peer. However there is data in this node or the subtree starting at this node. As soon as more peers are assigned to the corresponding path of the tree, random networks will be created for this node of the tree.

As mentioned before, the information displayed when selecting a node of the tree depends on the color of the node. The upper left of the Tree tab shows information about the responsibility of the peer for the selected node (Normal, Shanghaied, or Shortcut) and the address of the 3nuts peer and the random network peer beneath, if the selcted node of the tree is a path node (blue colored). If the node of the tree is a branch link (green colored) the age of the link to the other peer participating in the corresponding random network of the 3nuts node is shown.

For the load balancing mechanism the weight of nodes and subtrees play an important role. The weights themselves are calculated in the application running on top of the 3nuts network. Yet, the weights for the selected 3nuts node and its subtree obtained from the application are also shown. The weight of a path node has always a version tag. This is of major importance, since the nodes participating in a 3nuts node exchange the weights using randomized rumor spreading mechanisms.

The nodes of a 3nuts tree are represented by random networks. The peers participating in such a random network are all the peers, which have been assigned to that node of the tree by the load balancing mechanism. Using a double click on a path node (blue colored)

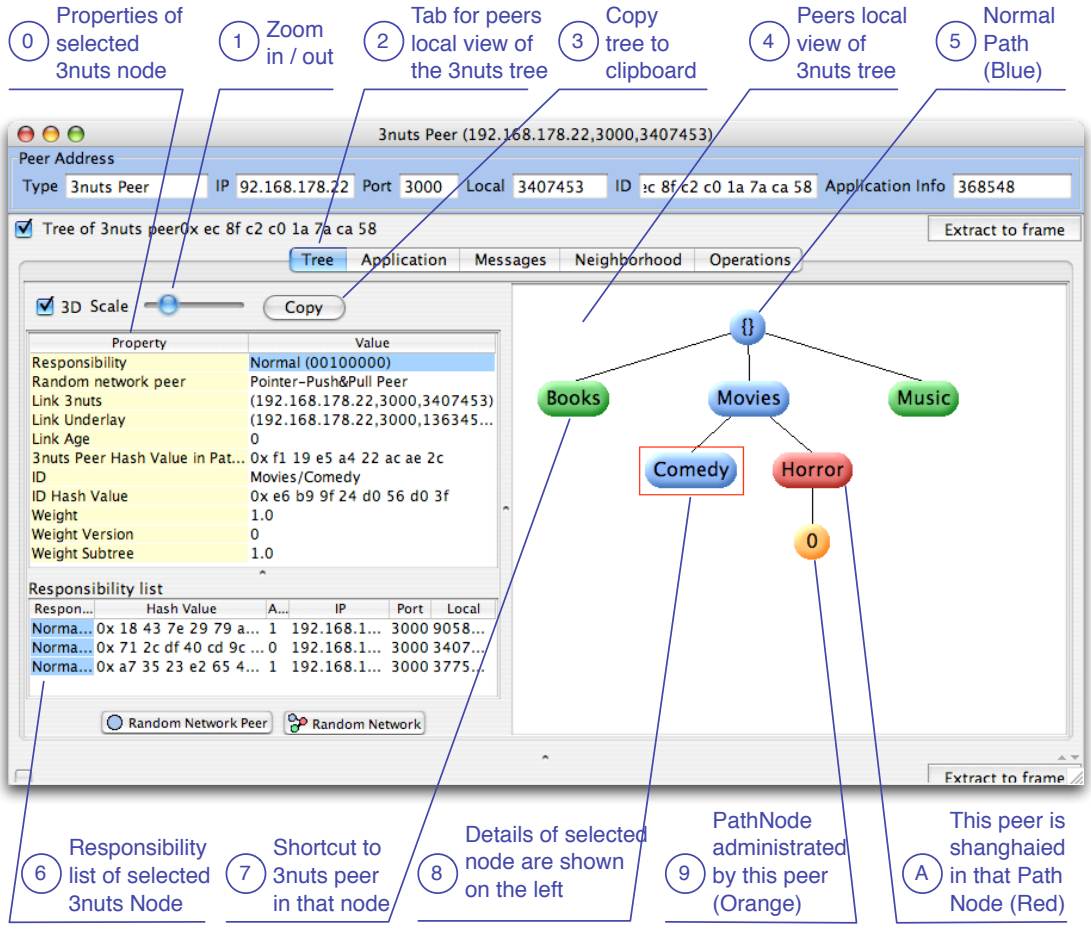


Figure 3.5: The tree view of a 3nuts peer.

a new window showing the corresponding random network will show up (see Figure 3.9). A detailed description of the contents of this window can be found in Chapter 3.5.

Note, that it is possible to navigate through the network using the displayed tree in the Tree tab: By double-clicking on a branch link (green node), the peer trace window will change to the peer of the branch link. So, all contents of the window now show information about the peer reached. This means in particular, that the peer address shown on top of the window and the local view of the 3nuts tree will change.

3.4.2 The Weighted Distributed Hash Table visualization

In 3nuts Weighted Distributed Hashing is used for load balancing, i.e. to assign peers to tree nodes. This hashing mechanism is performed in every node of the tree and may be visualized to analyze the behavior of a peer. To show the visualization, open the popup-menu by right clicking a blue colored node in the tree tab and select *WDHT (Double Hashing)*. A new window as shown in Figure 3.6 will be opened.

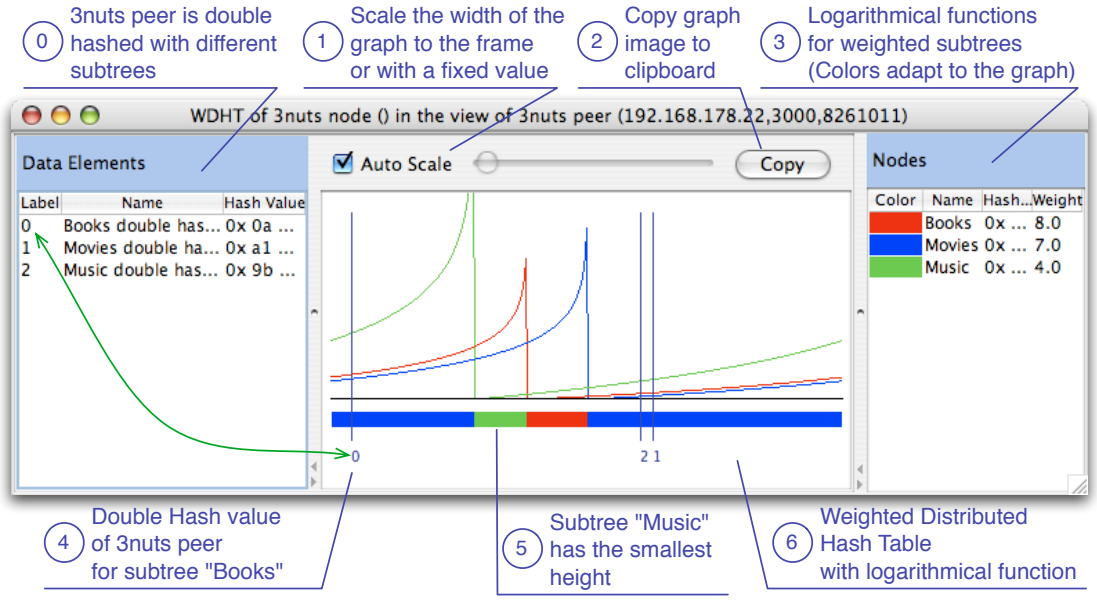


Figure 3.6: The Weighted Distributed Hash Table visualization

The table on the right side in the window shows the child nodes of the selected node (in Figure 3.6 these are the nodes *Books*, *Movies*, and *Music*). In the middle of the window the weighted distributed hash table (WDHT) is shown with weighted logarithmic

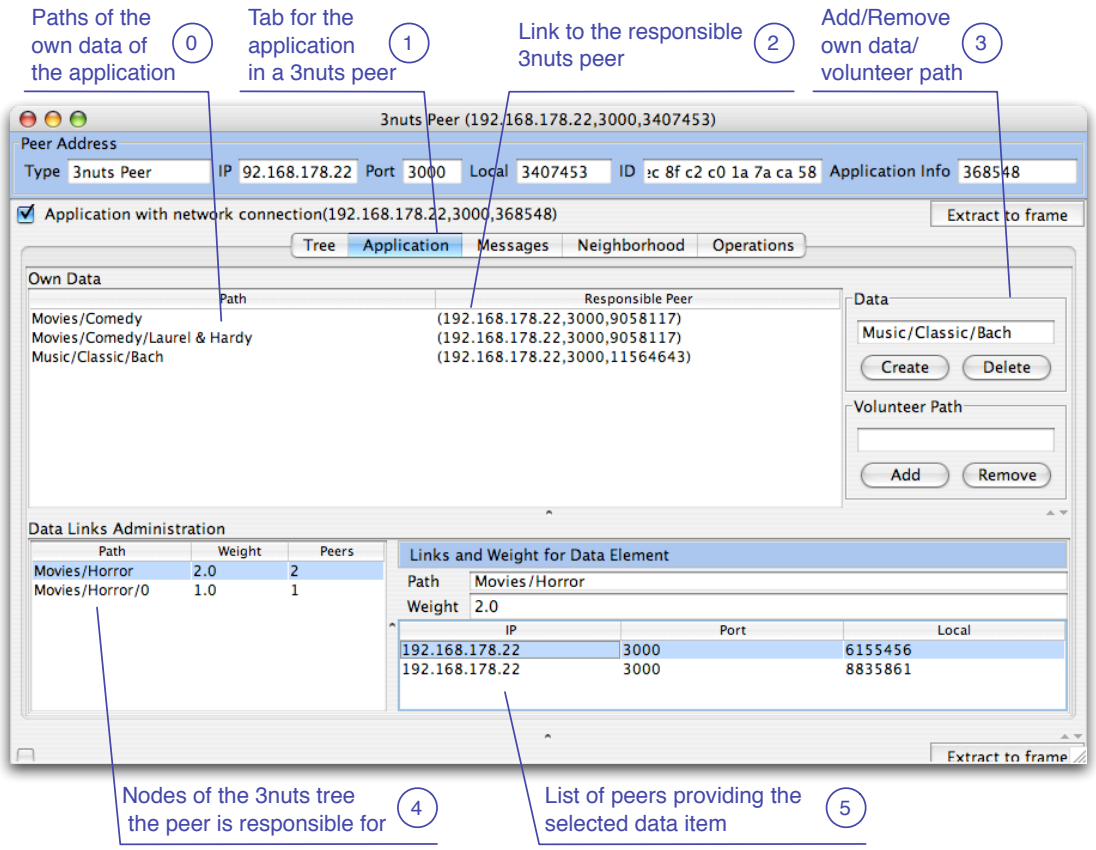


Figure 3.7: The Application tab.

functions of the child nodes and positions a peer has been hashed to (one position per child node). Using this WDHT, it is determined to which child node the peer will be assigned to — we will not go into further detail in this user guide.

3.4.3 The Application tab

As mentioned before, every peer of the 3nuts network is running a test application providing virtual data. This virtual data defines the tree structure the 3nuts network adapts to. Please see Chapter 3.3 for information on how to change the number of data elements generated at a peer, etc..

The Application tab (see Figure) is devided in an upper and lower half. The upper half shows information on the data elements generated by the application at this 3nuts peer.

Note that usually another peer of the 3nuts network will be responsible for these data items. In particular this will be the peer responsible for the node of the tree defined by the identifier of the data item. For every data item the responsible peer is shown next to it. On the very right side of the tab ③ new data items can be created or existing ones delteted by entering the identifier and clicking Add Data afterwards.

The lower half of the Application tab shows informations about data items the peer is responsible for ④, i.e. data items whose identifiers describe nodes of the 3nuts tree, where the peer is particpating as a path node and has highest responsible among all other peer participating in this node of the tree. For each of these data items the weight and number of peers providing it is shown. When one of the data items is selected, the addresses of the peers actually providing this data item is displayed on the right hand side ⑤.

3.5 Visualizing the 3nuts Network

The decentralized nature of a peer-to-peer network makes it hard to get a global view of the network. However, such a global view is extremely useful for checking the behavior of peers and debugging. Therefore the 3nuts software comes with a centralized debug tool. This tool can be started using the Show button in the 3nuts main window, see Figure 3.2. A screenshot is shown in Figure 3.8.

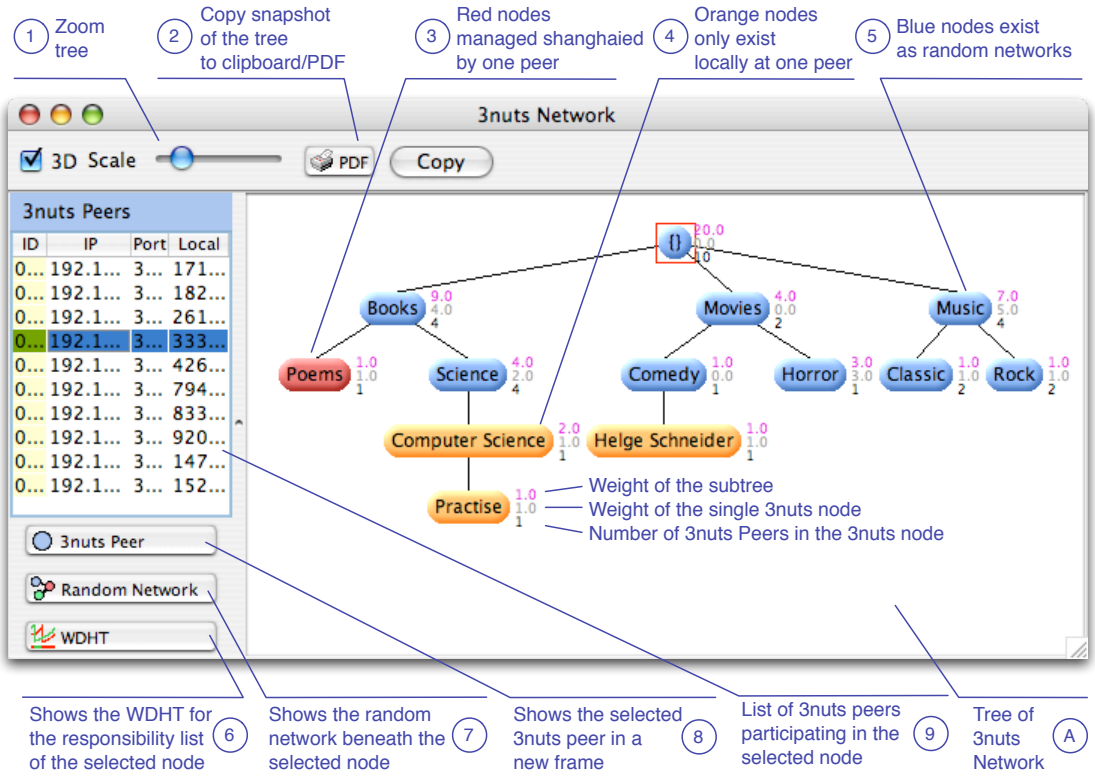


Figure 3.8: 3nuts network

The main part of the window shows the 3nuts network tree, defined by the data currently available in the network. The nodes of the tree are colored blue ⑤ if the node is actually represented by a random network. The orange portion ④ of the tree is currently not existing as a random network, yet these parts of the tree exist locally in the 3nuts peer, which has the highest responsibility in the first blue colored node above an orange colored subtree.

Red nodes ③ are nodes of the tree to which no peer has been assigned to by the load

balancing mechanism. In this case the peer that has the the highest responsibility in the parent node will be responsible for the red node until another peer is assigned to the red node.

The visualization of the tree can be adjusted in the following way: The nodes of the tree can be shown as rectangle or as shaded 3-dimensional ovals. Furthermore, you can zoom in and out using the slider ①. A snapshot of the current tree can be copied to the clipboard by pressing the copy button ②. The button PDF opens a dialog for saving the snapshot to a PDF file.

Clicking once on a blue node in the tree, i.e. a node with a random network beneath, all the 3nuts peers participating in the corresponding random network are shown in the list left to the tree visualization ⑦. Selecting a peer out of the list and then clicking the 3nuts Peer button ⑤ will open the 3nuts peer trace window of a peer.

The random network beneath a node of the tree will show up in a new window when double clicking on the corresponding node or selecting a node and then clicking the Random Network button ⑥. A screenshot of this window is shown in Figure 3.9.

In this window a list of all peers currently participating in this random network is shown on the left ⑤. The link structure of the random network is shown shown in the graph on the right ⑨. Since the random networks are maintained using Pointer-Push&Pull operations you will see how the links of the network change over time.

Again, there are some options for the visualization, which are almost equal to those in the centralized 3nuts debug window. There is just one additional check box ④ to toggle the visualization of self loops in the network.

If a fraction of random network peers for some reason got disconnected from the rest of the random network, the nodes of each connected component are shown in a different color.

Selecting a random network peer in the graph visualization or in the list, the routing table of that random network peer will be shown in the lower left of the window ⑦. Using a double click on a random network peer in the graph or in the list of peers or selecting a peer and clicking the Show Peer button ⑧, a new window with detailed information about this peer will be opened.

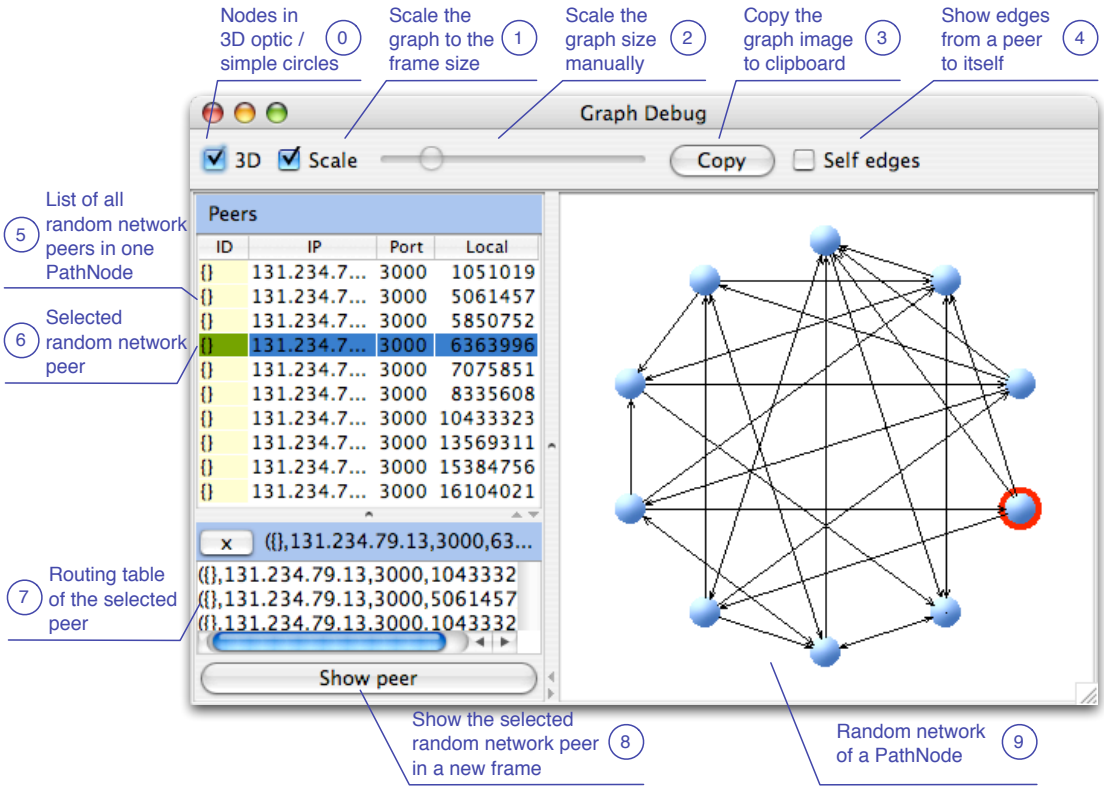


Figure 3.9: Random network connecting a PathNode in the 3nuts tree.

3.6 Measuring the Network Characteristics

3nuts includes tools for measuring the characteristics of a running 3nuts network. All measurements can be opened in the menu *Measurements* of the main-frame (see Figure 3.2). In particular, 3nuts allows to measure the distribution of the estimated tree weight (Figure 3.10), the distribution of the lookup hop distances (Figure 3.11), and the efficiency of the repair mechanism, i.e. the discovery of dead links using Pointer-Push&Pull operations (see Figure 3.12).

It is also possible to trace the communication of a 3nuts peers. The communication window (see Figure 3.13) can be opened by clicking on the button *Communication* in the second panel of the main frame (Figure 3.2). The image on the bottom of the window shows the communication state. The communication consists of an incoming channel on the left side (red) and an outgoing channel on the right side (green). These channels

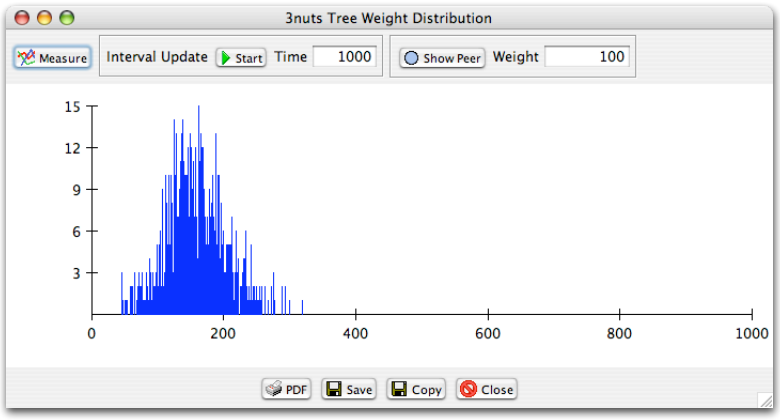


Figure 3.10: Load Balancing: Distribution of the estimated tree weight

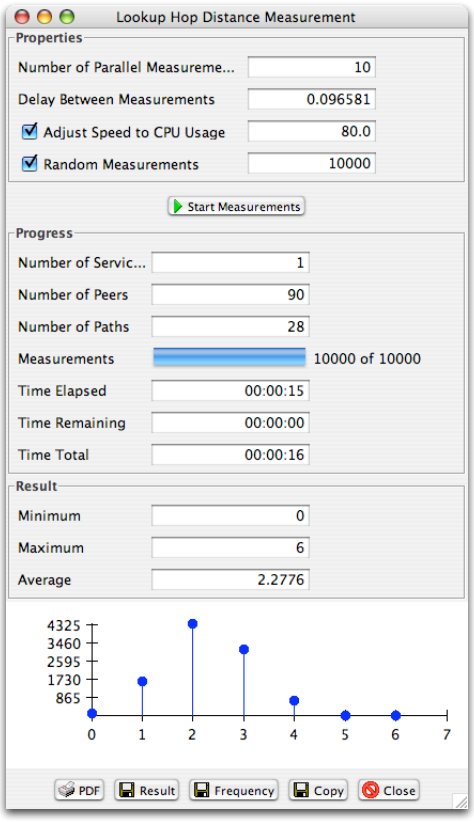


Figure 3.11: Measurement of the lookup hop distances in a 3nuts network

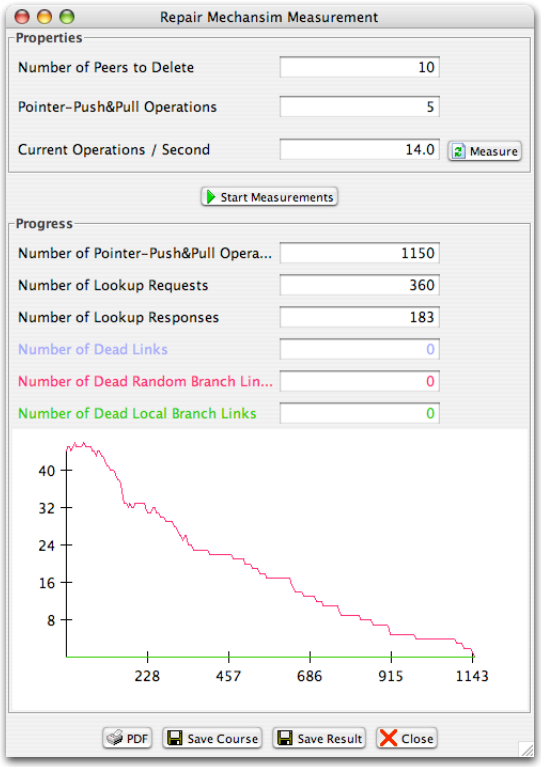


Figure 3.12: Measurement of the routing table repair in a network with removed peers

contain elements processing the messages. The peer itself is represent by the Participants block shown on top of the diagram.

The buttons on the upper section of the frame can be used to add or remove additional tools to the communication channels, e.g. by clicking on the button Message Utilization a new window is opened (figure 3.14) showing the number of exchanged messages per second for individual message types.

For the measurement two new channel elements are added to the Communication: In Message Utilization and Out Message Utilization (the image representing the Communication (3.13) will be updated).

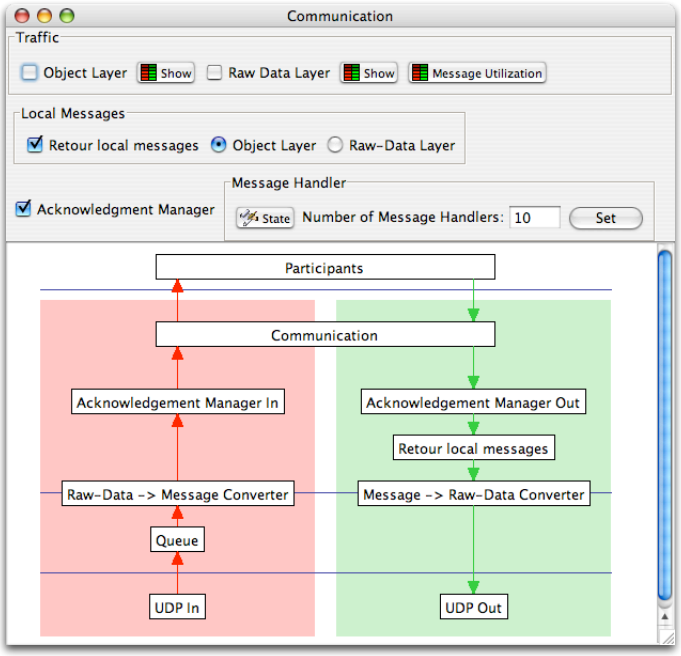
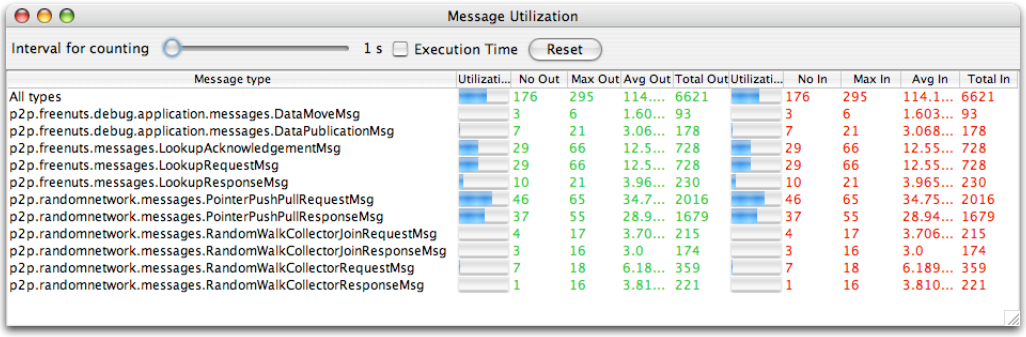


Figure 3.13: Communication framework for the peers



Message type	Utilizati...	No Out	Max Out	Avg Out	Total Out	Utilizati...	No In	Max In	Avg In	Total In
All types		176	295	114....	6621		176	295	114.1...	6621
p2p.freenuts.debug.application.messages.DataMoveMsg		3	6	1.60...	93		3	6	1.603...	93
p2p.freenuts.debug.application.messages.DataPublicationMsg		7	21	3.06...	178		7	21	3.068...	178
p2p.freenuts.messages.LookupAcknowledgementMsg		29	66	12.5...	728		29	66	12.55...	728
p2p.freenuts.messages.LookupRequestMsg		29	66	12.5...	728		29	66	12.55...	728
p2p.freenuts.messages.LookupResponseMsg		10	21	3.96...	230		10	21	3.965...	230
p2p.randomnetwork.messages.PointerPushPullRequestMsg		46	65	34.7...	2016		46	65	34.75...	2016
p2p.randomnetwork.messages.PointerPushPullResponseMsg		37	55	28.9...	1679		37	55	28.94...	1679
p2p.randomnetwork.messages.RandomWalkCollectorJoinRequestMsg		4	17	3.70...	215		4	17	3.706...	215
p2p.randomnetwork.messages.RandomWalkCollectorJoinResponseMsg		3	16	3.0	174		3	16	3.0	174
p2p.randomnetwork.messages.RandomWalkCollectorRequestMsg		7	18	6.18...	359		7	18	6.189...	359
p2p.randomnetwork.messages.RandomWalkCollectorResponseMsg		1	16	3.81...	221		1	16	3.810...	221

Figure 3.14: Message Utilization in the communication between the peers