OMOLSR User's Guide 1.0

Grup de recerca en Arquitectura i Serveis Telemàtics (AST)
Departament d'Enginyeria Informàtica i Matemàtiques
Universitat Rovira i Virgili
May 2008

Index

1 OMOLSR Basic functioning

OMOLSR is presented as a jar file named test_application.jar. This jar file is auto-executable and starts the class urv.emulator.AppLauncher. This class is the entry point both for the emulator and to run applications in a real environment. The behaviour of this class is specified in the ./conf/omolsr.properties file.
test_application.jar is available at http://ast-deim.urv.cat/popeye/test_application.jar

1.1 Emulator

When the jar file is launched as emulator, the AppLauncher creates as many instances of the application as nodes the emulation graph contains. Messages are not sent through the network, but through a special Emulation layer.

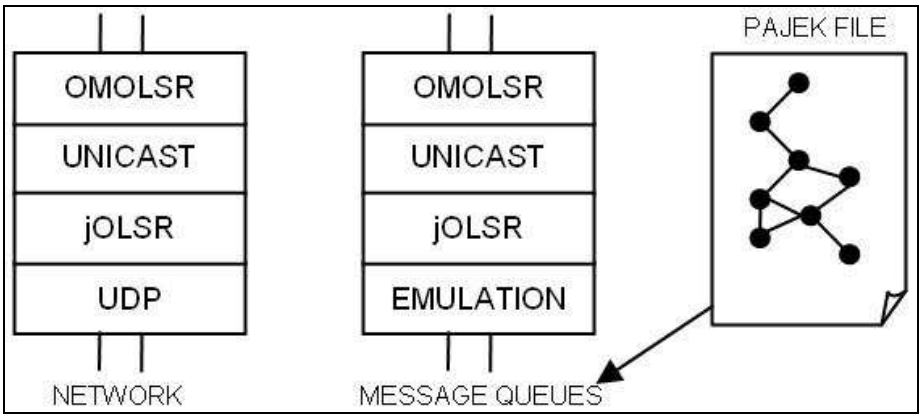

Fig. 1. JGRoups protocol stacks for real applications (left) and emulated applications (right). Messages from real applications are routed through the network, whereas messages from emulated applications are delivered considering the constraints in the graph file.

The emulator accepts task classes that support the emulation and collect statistics. Existing task classes are the following:

- CommunicationStatsTask: This task gathers information about all messages sent and received in the network by all applications. It verifies that all nodes that were in the view of the source node received the multicast message.
- MembershipStatsTask: This task gathers information about the groups created in the applications and the nodes that joined these groups. This information is checked with the view of each MChannel, in order to verify the correct behaviour of getView() method in the channel.
- TopologyChangesTask: This task is in charge of changing the underlying topology as it is defined in the network graph. This task also checks that the Neighbor table of the jOLSR protocol of each node is consistent with the real neighborhood of each node.

1.2 Real applications

When the jar file is launched in real environment, the AppLauncher only creates one instance of the application. Messages are sent using standard UDP sockets. Although tasks cannot be launched in real environment applications, a special flag (COMM_LOG) can be enabled to gather logs about sent and received messages.
There are several applications in the distributed jar file, mainly used for testing purposes. The most complete application is AppTest, that provides unicast and multicast

communication through an easy-to-use GUI. Another useful testing application is the SimpleSenderApplication. Under the package urv.app there are also other applications like MembeshipTest or RealTest.

1.2.1 AppTest

AppTest is a GUI application that was designed to test the protocols in real environments. This application allows the user to join different groups and start chats with all the group members. Chats can use multicast communication so every member in the group can read the messages, or unicast communication, that can only be read by the destination of the message.

Fig. 2 is a screenshot of this application. In the upper side the user can specify the multicast group to join. Once the user joins a group, a new big tab appears. In the left side of this tab there is the group members list and in the right side there is a graph representation of the network as seen by the local node. The local node is colored in green and the other nodes in red. There exists the possibility of changing the layout of the nodes by selecting the desired layout from the list below the graph. It is also possible to zoom in and out the graph representation.

The center panel holds the chat tabs. By double-clicking on a member of the list, a new chat tab for this member appears in the center panel. You can chat with different people of the same group and also in the default multicast chat. Messages sent to the multicast chat are received by all group members. When a chat message is received, the tab corresponding to the sender is highlighted in blue.

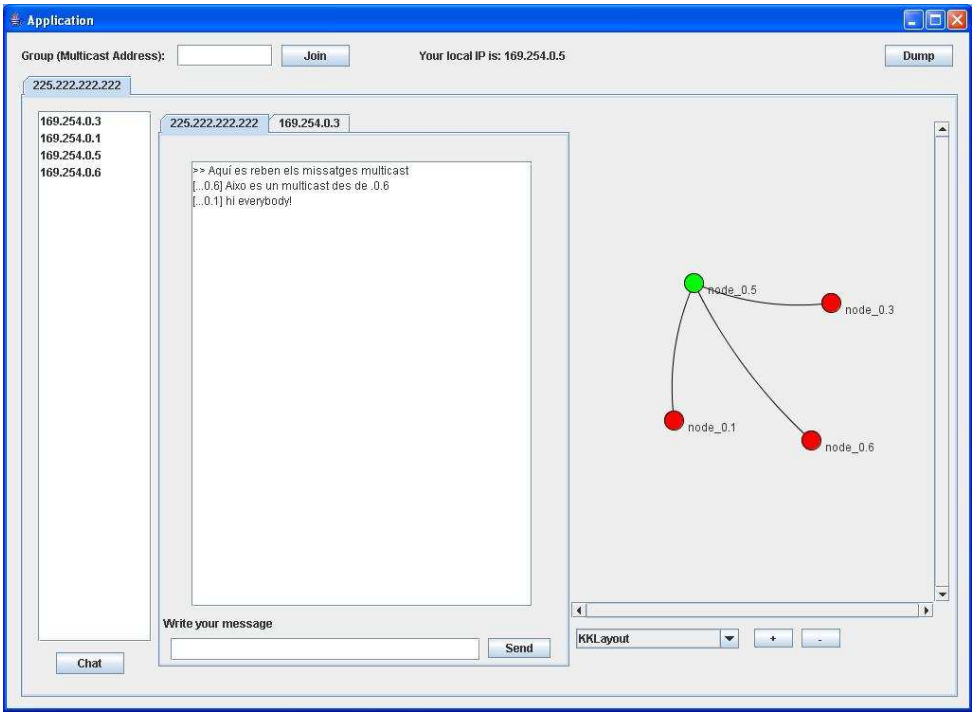

Fig. 2. Screenshot of AppTest application.

A button (Dump) in the top-left corner can be used to show the internal tables of jOLSR in run-time: Routing table, NeighborTable, TopologyTable and MprSet.

1.2.2 SimpleSenderApplication

SimpleSenderApplication was designed to test the protocols. This application basically creates a MChannel and sends multicast messages through it. If launched with the CommunicationStatsTask, you can see how many messages were correctly delivered to the other members of the group.

1.3 Properties file

The Java properties file that defines the behavior of OMOLSR is located at ./conf/omolsr.properties. The function of each property is described here:

- EMULATED: This flag indicates whether to perform network emulation or run a real application.
- COMM_LOG: This flag indicates if communication logging is enabled. This feature is only used for non-emulated environments.
- UNICAST_PORT: The unicast UDP port used in real applications.
- RELIABILITY: This flag indicates whether the protocol stack includes reliability protocols or not.
- MULTICAST_PROTOCOL: The protocol that will be used to send multicast messages. Currently, there are two protocols available: OMOLSR and SMCAST.
- GRAPH_FILE: The pajek file used for the network simulation. One application will be launched for each node in the network graph. The file can be static (with extension .net) or dynamic (with extension .tim). Dynamic graphs involve adding and hiding edges at different time instants.
- APPLICATION: The application class that will be launched, either in emulation or in real environment. Most applications can be used in both environments.
- EMULATION TASK: A list of tasks classes. These tasks are launched to support the emulation.
- SENDING_PROB: jOLSR control messages are sent with this probability. If the probability is 1.0, all messages will be delivered to the destination. If the probability is 0.0, all sent messages are discarded.

The possible values of the properties are listed in the following table:

Property name	**Possible values**
EMULATED	TRUE, FALSE
COMM_LOG	TRUE, FALSE
UNICAST_PORT	A port number > 1024
RELIABILITY	TRUE, FALSE
MULTICAST_PROTOCOL	OMOLSR, SMCAST
GRAPH_FILE	A graph file (extension .net or .tim)
APPLICATION	The fully qualified class name of an application class
EMULATION_TASKS	A comma separated values list of tasks classes
SENDING_PROB	A float value between 0.0 and 1.0

2 Creating new applications

In order to create new applications you must follow these guidelines:

- The application class must extend urv.emulator.core.Application. E.g.:

```
import urv.emulator.core.Application;

public class AppTest extends Application{
     ...
}
```

- The application class must implement the abstract method Application.start(). After the invocation of this method, the new application can start creating MChannels.
- The creation of the MChannel must be done with the method Application.createMChannel(MulticastAddress). Before sending and receiving messages, the application must invoke the method start on the created MChannel. E.g.:

```
MulticastAddress mcastAddr = new MulticastAddress();
mcastAddr.setValue("224.0.0.10");

MChannel mChannel = super.createMChannel(mcastAddr);
mChannel.registerMessageListener("MyApp",...);
mChannel.addMembershipListener("MyApp");
//We must call start before sending and receiving messages
mChannel.start();
```