This is an unfinished draft. If you have comments or corrections, please mark this document up and send it to jorg@ieee.org. If you find discrepancies between this document and the most recent version of the HyperCast software, please send a description of the problem.

Thank you,

Jörg Liebeherr

CHAPTER 11 Lotos – The Logical Topology Simulator

ABSTRACT

The Lotos simulator has an interactive GUI for simulating the overlay protocols of *HyperCast*. Lotos serves several goals: (1) It illustrates the operations of available overlay protocols and security schemes; and (2) It can be used to develop and debug new overlay protocols. The latter is possible because Lotos accesses the implementation of overlay protocols from the HyperCast code distribution.
Based on LotosUserManual (June 2005).

11.1. OVERVIEW

Assuming that Lotos is installed and the environment variables are set accordingly (see Appendix), Lotos is started with the command

```
>java lotos.GUI
```

The command displays a graphical user interface as shown in Figure 1.

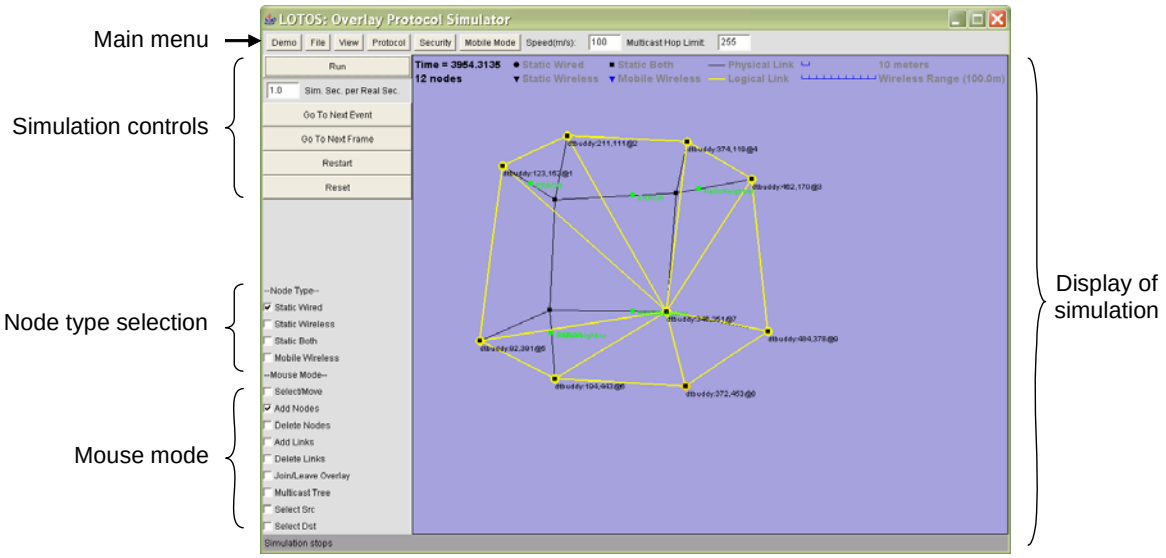

Figure 1. User interface of Lotos.

Figure 1 displays a simulation in progress in the simulation window. Nodes and links of the substrate network are shown as black points and lines. Nodes and links of the overlay network are shown as yellow nodes and links. Packets in transmission are shown as green squares. All nodes and packets are annotated with relevant information. Additional information can be obtained by accessing the statistics of the overlay nodes.

Describe the window….

- It is possible to run multiple protocols at the same time. However, each node can only run one node at the same time.
- Wireline link are full-duplex (propagation speed?)
- Wireless links use a simple media access control for transmission. Collisions may occur.
- Only routing protocols are simulated. There is no simulation of transmission of application messages.
- Propagation and transmission delays
- Unicast and multicast routing in the substrat network
- Broadcast in the substrate network
- Lotos can comfortably simulate twenty nodes. When the number is increased it consumes a signifnicatn amount of computing rsources. The resource consumption is increased when nodes are mobile and when security schemes are enabled.
- Lotos is specific to the Java implementation of HyperCast
- Lotos assumes that all required files are available in the directory where Lotos is started. If the file is not found, then Lotos accesses default files distributed with

the lotos.jar archive.

11.2. EXERCISES WITH LOTOS

The following exercises provide an overview of the basic functions of the Lotos user interface.

Exercise 1: Running a Demo simulation.

1. Click on the *Demo* button in the main menu and select one of the topologies. Then push the Run/Pause button to stop and resume the simulations. Observe the transmission of packets (shown in green).
2. Slow the rate of the simulation by setting the value of the *simulation seconds per real seconds* to 0.02. The change takes effect when you hit *Return*.
3. Enable and disable the various display options in the *View* pull down menu.
4. Pause the simulation and advance by single steps by pushing the *Next Event button.*
5. Restart the simulation by pushing the *Restart* button.

Exercise 2: Building a simulation with wireline nodes.

1. Reset Lotos to a base state by pushing the *Reset* button.
2. Select the node type *static wired*.
3. Select the *Add nodes* radio button. Click (with the left mouse button) in the blue window and add a few nodes.
   Note: When Lotos is started no protocol is selected. Thus, when creating new nodes, only substrate network nodes are created without an overlay protocol.
4. Go to the *Protocol* pull down menu and select the SPT (Spanning Tree) protocol.
5. Select the *Add nodes* radio button.
6. Click (with the left mouse button) in the blue window and add a few nodes. Each node creates an underlay network node and a node of the SPT protocol. *Note:* Now, each added node consists of a substrate node (shown in black) and an overlay node (shown in yellow)
7. Select the *Add links* button.
8. Create substrate network links by clicking on one node and drag the mouse to another node. Set enough links so that all nodes are connected by a path in the substrate network.
9. Start the simulation.
10. Select the *Select/Move* button. Click in the upper left corner of the blue window and drag the mouse to the lower right corner of the window. Nodes that are selected have a red circle around them. Select all nodes.
11. Change the overlay protocol by selecting the *DTBuddyList* protocol from the *Protocol* pull down menu.
12. Add new nodes and links while the simulation is running.
13. Deactivate an overlay node by selecting the *Join/Leave Overlay* button and

clicking on the button.

Exercise 3: Building a simulation with wireless nodes.

1. Reset Lotos to a base state by pushing the *Reset* button.
2. Go to the *Protocol* pull down menu and select the SPT (Spanning Tree) protocol.
3. Select the node type *static wireless.*
4. Start the simulation.
5. Select the *Add nodes* radio button and add a few nodes. The wireless communication range is indicated by circles.
6. Select the node type *mobile wireless* and leave the *Add nodes* radio button and add a few mobile nodes.

Exercise 4: Saving and loading topologies.

1. Pause an ongoing simulation from Exercise 1-3.
2. Select *File*→Save *All* in the toolbar. Provide a file name and save the file. This saves the substrate network topology, the overlay topology and type of overlay network as an XML file.
3. Push the *Reset* button to delete the network and reset the simulation.
4. Select *File*→Open *All* in the toolbar. Open the XML file that was saved in Step 2.
5. Click on *Start* to continue the simulation.

Exercise 5: Accessing information about a node.

1. Start with an ongoing simulation from Exercises 1-4.
2. With the mouse, right-click on an overlay node in the simulation window, and select *Node History.* This displays recent events at this node, such as timeouts, and the transmission and reception of message. Close the window.
3. With the mouse, right-click on an overlay node in the simulation window, and select *Node Statistics.* In the top window type the string /Node/ and push the *getStats* button. This displays the current value of statistics of the node (see Chapter "Monitor and Control"). Close the window.

11.3. GRAPHICAL INTERFACE

We next provide a description of the graphical user interface.

Toolbar

The toolbar is shown in Figure 2.

Figure 2. Toolbar.

The toolbar includes a series of buttons, that each display a pull down menu when selected. The *Speed* field sets the speed of all mobile nodes in terms of *meters per second*. The *Multicast Hop Limit* field sets the range of broadcast messages in the

underlay network, in terms of number of hops traversed in the underlay network before a message is dropped. The buttons function as follows.

Demo: Displays a list of pre-configured network topologies. The demo button gives the ability to quickly run simulations. A simulation is run by selection a topology from the pull down menu and then selecting *Start*. See Section 2 for customizing the pull down menu menu.

File: Deals with saving and loading the substrate and logical network setting displayed in the simulation window. The user can choose to save/load only the substrate or logical network, or both. The network is saved to an XML file with a user specified name. The button is not displayed when the program is run as an Applet.

View: Sets display options of the overlay network. A user can choose to zoom in, zoom out or scale the network to the size of the simulation window. Also, it is possible to select the displayed items in the simulation menu, by toggling checkboxes.

Protocol: Displays a list of available overlay protocols that can be simulated. When a protocol on this menu is checked, the next node that is created or activated will run the selected protocol. If some nodes are selected using the *Select/Move* button, the selected nodes (indicated by a red circle) change to the selected protocol. See Section 2.1 for information to customize this menu.
Note: Initially, when Lotos is started, no protocol is selected.

Security: Displays security modes that can be simulated by Lotos. When a security setting on this menu is checked, the next node that is created or activated will run the selected security setting. If some nodes are selected using the *Select/Move* button, the selected nodes (indicated by a red circle) change to the selected security setting See Section 2.1 for customizing this menu.

Mobility Mode: Chooses the mobility mode of mobile wireless nodes added to the network. Currently there is only one mobility mode, the *Linear Motion* mode. Here, a node randomly selects a position on the border of the simulation window and moves to the position at a constant speed. If the border is reached a new position is selected.

Simulation controls

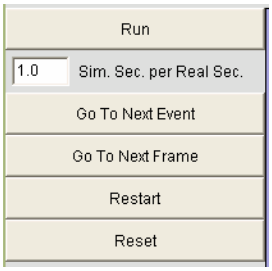

Figure 3. Simulation controls

The buttons control the progress of the simulation are shown in Figure 3, and perform the following functions:

Run/Pause: Starts or pauses the simulation.

Simulation speed: Sets the speed of the simulation in terms of simulated second per second of wall clock time.

Go To Next Event: Runs the simulation until the next event, e.g., until a message is sent or received.

Go To Next Frame: Moves the simulation ahead by one displayed frame.

Restart: Resets the simulation clock to 0, and resets all protocol nodes. The substrate network is not modified.

Reset: Resets simulation clock to 0 and clears the substrate network and overlay nodes in the simulation window.

Node type selection

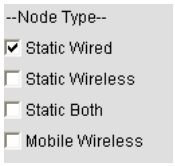

Figure 4. Node type selection

The checkboxes to select the type of substrate node added to the network are shown in Figure 4. When a new node is created (with the *Add Nodes)* function, selected node type is created. The following settings are available:

Static Wired: Selects a wireline node. A wireline node must be connected to other wireline nodes by a link (as is done with the *Add Link* function).

Static Wireless: Selects a non-mobile node that uses wireless transmissions. The range of transmissions is graphically indicated by circles.

Static Both: A node that transmits and receives over a wireless interface as well as over wireline links. A node of this type can act as a bridge between wireless and wireline nodes.

Mobile Wireless: A node that transmits over a wireless interface and is mobile following the mobility mode and speed specified in the toolbar.

Mouse mode

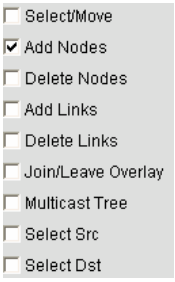

Figure 5. Mouse mode.

The mouse mode determines the operations performed when clicking the left mouse button in the blue simulation window. The available mouse modes are shown in Figure 5 and with the following results:

Select/Move: Clicking on a node selects a node. A selected node is indicated by a red circle. By holding the Control key multiple nodes can be selected. All nodes in a rectangular area can be selected by clicking in the upper left corner of the area and dragging the mouse to the lower right corner of the area. Selected nodes can be moved in the simulation window by clicking on one of the selected nodes and dragging the mouse to the new position. The protocol and security settings of selected nodes is changed by selecting a protocol and a security mode from the toolbar.

Add Nodes: When a user clicks in the simulation window, a new substrate node will be added at that position. If a protocol has been selected in the protocol menu, a protocol node is activated on the created substrate node.

Delete Nodes: Clicking on a node in the simulation node removes the substrate node and, if activated, the protocol node.

Add Links: A substrate network link between two nodes is created by clicking on one node and dragging the mouse to the other node. Both nodes must have a wired interface, that is, they must be either of type *Static Wired* or *Static Both.*

Delete Links: Clicking on a substrate network link removes the link.

Join/Leave Overlay: Clicking on a node toggles the activation and deactivation of the overlay protocol. This is equivalent to joining and leaving the overlay network. If an overlay protocol is running on the substrate node, the node is deactivated. If no overlay protocol is running on the substrate node, the overlay protocol selected in the toolbar is activated on this node.

Multicast Tree: Clicking on an activated node displays the multicast distribution tree that has the selected node as the root of the tree. The displayed tree illustrates how multicast application messages from the selected node will be forwarded in the overlay network.

Select Src: Clicking on a protocol node to choose the source of a unicast path. If both source and destination is selected, a unicast route between them is displayed if it is available.

Select Dst: Click on a protocol node to choose the destination of a unicast path. If both source and destination is selected, a unicast route between them is displayed if it is available.

Popup menu of nodes

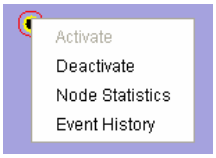

Figure 6. Popup menu.

Whenever user clicks with the right mouse button on a node in the simulation window, a popup menu as shown in Figure 6 is displayed. The selections are as follows:

Activate: If there is no protocol running, selecting this item adds (activates) a protocol node on the substrate node using the protocol selected in the protocol menu.

Deactivate: Removes **(**deactivates) a protocol node if an overlay node is activated.

Node Statistics: Displays a dialog box that accesses the statistics of the protocol node. Statistics in HyperCast are organized as XML documents and are accessed with XPath expressions. Refer to Chapter “Monitor and Control” for a discussion of the statistics in HyperCast. Figure 7 illustrates the dialog box. The user types in an XPath expression in the top line (e.g., /Node/NodeAdapter in Figure 7), and then selects one of the buttons at the bottom of the box. The results of the access to statistics are displayed in the center of the dialogue box. The buttons perform the following functions:

- **getStats** – Reads a statistic indicated by the provided XPath expression and displays the result as an XML document. The center box in Figure 7 shows the

results of the *getStats* for the XPath expression */Node/NodeAdapter*. These are the statistics of the node adapter of the selected node.

- **setStats** – Changes values of a statistic. This command assumes that there is an XPath indicating the value to be changed and an XML element that indicates the new value. For example, the string
  /Node/LogicalAddress <LogicalAddress>1234<LogicalAddress>
  changes the logical address to *1234.*
- **getReadSchema** – Displays an XML schema that shows the structure and the format of statistics that can be read by a *getStats* invocation.
- **getReadSchema** – Displays an XML schema that shows the structure and the format of statistics that can be changed by a *setStats* invocation.
- **close** – Closes the dialogue box.

Event History: Displays a window that shows the recent event history of the selected protocol node. The events recorded include timer events, messages sent, and messages received. The window displays the 100 most recent events of the node. The event history window of a node is shown in Figure 8. The figure displays three smaller windows. The window on the left shows a summary of the recorded events, including the time of the event and a description of the event type. Selecting one of these events displays the details of the event in the upper box on the right, including the source and destination address and the message, as well as a hex dump of the complete message. The lower window on the right displays log messages generated by this node.

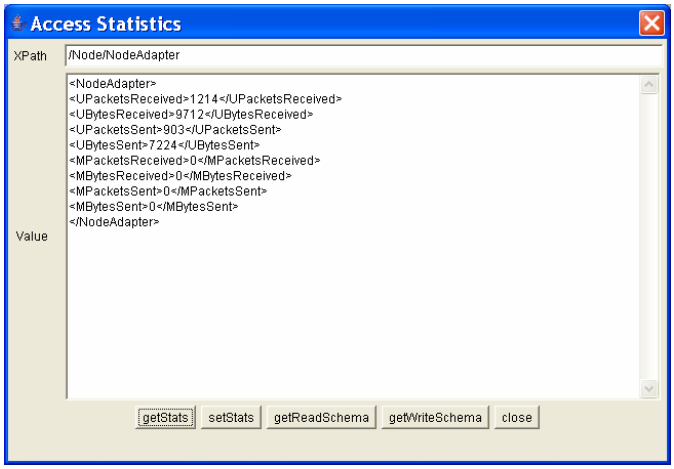

Figure 7. Dialog box for node statistics.

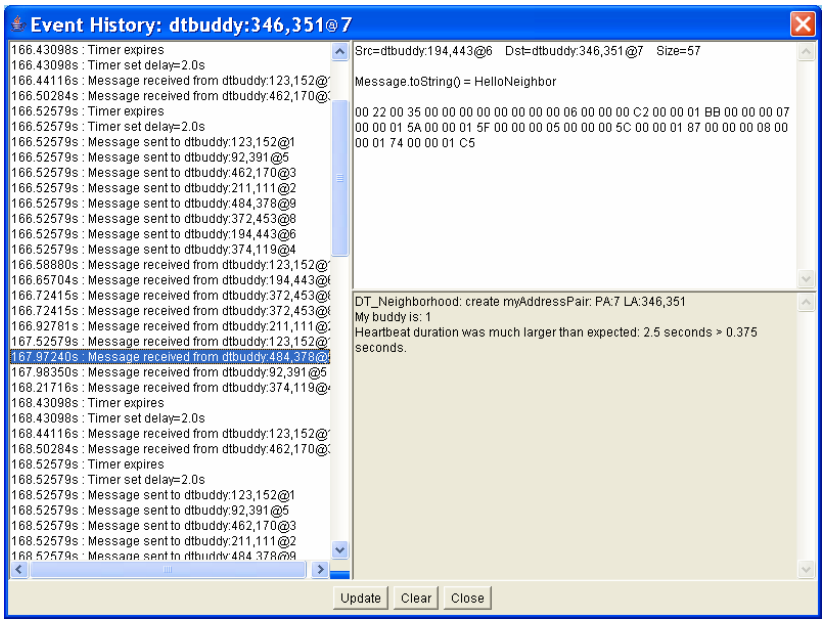

Figure 8. Event history.

11.4. CONFIGURATION FILES

Lotos uses three XML configuration file. The main configuration file is *SimHypercast.xml* which contains all HyperCast configuration attributes for overlay protocols and security protocols. The other files are *DemoMenuConfig.xml* and *demo1.xml*, which set up the network configurations available through the *Demo* button in the toolbar.

The structure of the file *SimHypercast.xml* is shown below. The file contains a list of *SecuritySetting* and *HyperCastProtocol* elements. The *SecuritySetting* elements each specify a security scheme. The value of the *name* attribute is identical to the element name of the corresponding property in the configuration file for overlay sockets, and the content of the *Security* element is valid with respect to the XML schema file *hypercast.xsd* for overlay sockets. In other words, the content of the *Security* element is compatible with the *Security* element in configuration files for overlay sockets. The elements *PrivateProperty* declares the content of the private properties needed by the security schemes (e.g., passwords). In the overlay socket configurations, private properties contain secret information which is never stored in a configuration file or transmitted to other overlay socket. Instead, private properties must be set by the application program. In the Lotos simulation, the private properties are obtained for a file.

The *HyperCastProtocol* elements specify overlay protocols. The *name* attribute contains the name of the protocol that is used in the *Protocol* button to identify the protocol. The element *AgentClass* specifies the name of a Lotos class that is responsible for displaying entities and events related to an overlay protocol in the simulation window. This class knows the type of *I_Node* class that implements protocol. The *Node* element specifies the properties of the overlay protocol. The element is identical to the node element of the same protocol in the configuration file for overlay sockets.

```
<SimHypercast xmlns=http://www.cs.virginia.edu/hypercast
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.cs.virginia.edu/lotos.xsd">
<SecuritySetting name="NeighborhoodKey1">
```



```
   <Properties>
      <Public>
         <Security>
            <!-- public properties of NeighborhoodKey1 scheme -->
            ...
         </Security>
      </Public>
   </Properties>
   <PrivateProperty XPath="/Private/KeyStorePassword" value="..." />
   <!—- private properties for NeighborhoodKey1 scheme -->
   ...
</SecuritySetting>
<SecuritySetting name="...">
   <!-- properties of additional security scheme -->
   ...
</SecuritySetting>
<HypercastProtocol name="SPT">
<AgentClass>SPTAgent</AgentClass>
   <Properties>
      <Public>
         <Node>
            <SPT>
               <!-- public properties for the SPT protocol -->
               ...
            </SPT>
         </Node>
      </Public>
   </Properties>
   </HypercastProtocol>
   <HypercastProtocol name="...">
      <!-- properties of additional protocols  -->
   </HypercastProtocol>
```

All nodes that are created with a selected overlay protocol and security scheme use the same configuration parameter. For overlay protocols it is possible to have multiple instantiations of the same protocol with different configuration parameters. This is done by declaring multiple *HyperCastProtocol* elements with identical *AgentClass* values (so that they instantiate the same type of protocol) but different name attributes (so that they appear as different choices in the Protocol pull down menu). However, different protocol settings need to be given different names in order to be distinguishable in the *Protocol* menu.

11.5. LOTOS CUSTOMIZATION

Lotos can be customized to add new overlay protocols and new security schemes that were programmed for HyperCast. It is also possible to add network configurations to the *Demo* menu in the toolbar. The customizations are done without changing existing Lotos or HyperCast source files.

Adding a new protocol

All overlay protocols simulated by Lotos must implement the *I_Node* interface. Thus, a each overlay protocol has the same requirements as a protocol running in an overlay socket. In addition, there are a few rules that the protocol implementation must obey to be correctly simulated by Lotos:

1. The finite state machine which executes the protocol must be initialized and started, only after the *joinOverlay()* method is called. Particularly, the constructor of the *I_Node* protocol node should not initialize.
2. After the *leaveOverlay()* method is called, the protocol node must stop processing handling timers and messages. Also, pending timer events must be cleaned up (removed) after this method is called.
3. The protocol node should not assume that there are separate threads for timers and messages. As a result, using *wait()* or *notify()* functions to perform synchronization between processing of messages and timers is inappropriate.
4. All protocol messages that are processed by the *I_Node* object should implement the *toString()* method to return a short label for a message, which is displayed in the simulation window when the message is transmitted.

If an *I_Node* protocol implementation adheres to these, running the protocol in Lotos requires to write one class, referred to as agent class, and edit one of the configuration files.

Create Agent class

The first step of adding a new protocol to the simulator is to create a class implementing *I_SimHyperCastAgent* interface. This class works as a bridge between a HyperCast protocol node (of type *I_Node*) and the simulation. The definition and description of this interface are shown below.

An agent class deals with creating new *I_Node* objects to be simulated in Lotos. The servers that need to be run to support the protocol should also be implemented as an *I_Node* class, and the server objects should be created before any other normal protocol *I_Node* is created (see *createNode()* and *needServer()* methods). An agent class should store the server objects that was created in order to check if they are removed in the implementation of *nodeRemoved()* method.

An agent class also tells the GUI interface how a protocol node is displayed on the GUI window using *getColorForNode()*, *getShapeForNode()* and *getLabelForNode()* methods. The *getInfoForNode()* method generates a text string that is saved with a protocol node to a file for later restoration. The *info* string should contain all information necessary to restore the protocol node later in the same agent object.

The following is a discussion of the methods in the *I_SimHyperCastAgent* interface:

public String getName()

Returns the name of this protocol **(Is this the name of the class?)**. This name will work as an identifier of the protocol. However, the name is not the text that is shown in the protocol menu since several protocol menu items might share the same agent class.

public void setNetwork(simHCNetwork network)

Makes the simulated network available to the agent class. This is needed for

accessing global information about the information, e.g., provide access to tall nodes in the network.

public I_Node createNode(XYAddress coords, HyperCastConfig config, I_UnicastAdapter adapter, String info)

This function creates a new overlay protocol node of type *I_Node* and adds it to the simulated network. The created I_Node object is put on an existing physical node on the simulator GUI interface. >>>> If *needServers()* is true, then this function creates a server node that is needed to run the protocol. ← ????

The arguments are as follows: *coords* is the position for this node in the simulation window; *info* is used to restore a saved protocol node from a file. This argument can eb *null*. The value of *info* was originally obtained by calling the *getInfoforNode()* method; *adapter* is the node adapter to be used by the new node.

public boolean needServers(HyperCastConfig config)

This function should return true if the protocol requires a rendezvous server (which is not serving as protocol node). In this case, the rendezvous server must be created before any other protocol node. The servers will be created one by one by calling the *createNode()* method. An example of a protocol that needs a server is the *DTServer* protocol.

public String getInfoForNode(I_Node node)

The string is used to restore a saved protocol node from a file. **What is the content?**

public String [] getMenuItems()

Returns a list of menu item labels for this protocol if it has need to define protocol dependent operations for protocol nodes created by this agent class. There should not be duplicate menu names in this list. . **Give an example.**

public void menuFired(String menu)

This method defines the operations associated with the popup menu items defined by *getMenuItems()*. The method fires the menu item using the label name of the item.

public Color getColorForNode(I_Node node)

Returns the color for an I_Node object. The returned color should depend on the current state of the overlay protocol node.

public int getShapeForNode(I_Node node)

Returns the shape of an *I_Node* object. The resulting shape should depends on the type of the node.

public String getLabelForNode(I_Node node)

Displays the label associated with an *I_Node*. Get the **display** label for a node.

public void nodeRemoved(I_Node node)

The method is called when a protocol node is removed in Lotos, e.g., by deleting a node. The method tells the agent when a protocol node is removed from the overlay network so that the agent can perform necessary cleanup functions. For example, in the *DTServer* protocol, when the DT server is removed, the agent is informed by this function and will create another DT Server the next time the *createNode()* method is called.

Modify configuration files

After an agent class is created, the next step of adding a protocol into *Lotos* is the addition of a *HyperCastProtocol* element to the configuration file *SimHypercast.xml*.

Suppose we written a new protocol in terms of an *I_Node* class with name *MyProtocol*. For *Lotos*, we create an agent class with name *MyProtocolAgent*, with the following structure:

```
<SimHypercast>
   ... ...
   <!-- the name attribute will appear in the protocol menu -->
   <HypercastProtocol name="MyOwnProtocol">
      <!-- the name of the agent class -->
      <AgentClass>MyProtocolAgent</AgentClass>
      <Properties>
         <Public>
            <Node>
               <MyProtocol>
               <!-- the properties of the MyProtocol node -->
               ... ...
               </Protocol>
            </Node>
         </Public>
      </Properties>
   </HypercastProtocol>
   ... ...
</SimHypercast>
```

The *name* attribute in the *HypercastProtocol* element is the name of the item in the *Protocol* pull down menu. The *AgentClass* specifies the name of the agent class that mediates between *Lotos* and the protocol node objects. The configuration attributes for the protocol nodes are provided in the *Properties* element.

Adding a new security scheme

Security settings can be added by adding a *SecuritySetting* element to the *SimHypercast.xml* file. Again, the name attribute declares how the scheme is presented in the pull down menu of *Lotos*. The implementation of the security scheme is determined by the selection of the *KeyModeMethod*. Below is an example for adding a new security scheme with name *MySecurityScheme* uses a (fictitious) key mode method with name *NewKeyMode:*

```
<SimHypercast>
   <!-- the name field will appear in the toolbar menu -->
   <SecuritySetting name="MySecurityScheme">
      <Properties>
         <Public>
            <Security>
            <!-- below are the public properties for the scheme -->
            <KeyModeMethod>NewKeyMode</KeyModeMethod>
            <SecurityLevel>privacy</SecurityLevel>
            <CertificateLocation>testcert.cer</CertificateLocation>
               ... ...
            </Security>
```



```
         </Public>
      </Properties>
      <!-- below are the private properties for the scheme-->
      <PrivateProperty XPath="/Private/KeyStorePassword"
                                                value="password"/>
      <PrivateProperty XPath="/Private/PrivateKeyAlias"
                                                value="testpair"/>
      <PrivateProperty XPath="/Private/PrivateKeyPassword"
                                                value="password"/>
      <PrivateProperty XPath="/Private/GroupKey"
                                          value="1234567812345678"/>
   </SecuritySetting>
</SimHypercast>
```

Adding a new demo network

Adding a new network to the Demo button in the toolbar is done in two steps. First, manually draw a new network with substrate nodes and overlay nodes in the simulation window, save the network by selecting *File→Save All* from the toolbar menu, and type a file name, e.g., *mynewdemo.xml*. In the second step, edit the file *DemoMenuConfig.xml* and add the following item:

```
<DemoMenu>
    <!-- the name field will become a item in the Demo menu -->
   <Demo name="MyDemo">
      <FileName>mynewdemo.xml</FileName>
   </Demo>
</DemoMenu>
```

If Lotos is restarted, the new network is available from the *Demo* pull down menu under the name *MyDemo*.

11.6. APPENDIX: LOTOS INSTALLATION

System requirements

The HyperCast Simulator requires Java version 1.4.x or higher. The system must have at least 30MB free memory and support graphic display.

Installing ZIP archive with binaries

Download a ZIP archive with name

lotos-xxx-yyyymmdd.zip

where *xxx* is a tag that identifies where the archive was created, and *yyyymmdd* denote the year, month, and day when the archive was created. Copy the file to a directory where you want to run Lotos. Extract ('unzip') the archive with a utility such as WinZip (on Windows) or the unzip command (on Unix or Cygwin).

The content of the archive

The contents of the ZIP archive is shown in Table 1.

Table 1.

Libraries	
lib/lotos.jar	HyperCast Lotos simulation software
lib/hypercast.jar	HyperCast software
lib/hypercast-pastry.jar	HyperCast wrapper of FreePastry.
lib/bcprov-jdk14-122.jar	Free Java implementation of cryptographic algorithms (www.bouncycastle.org)
lib/xalan.jar	XSLT processor for processing XML documents
lib/lotos.html	HTML file that starts the Lotos Applet
XML files	
SimHypercast.xml	Configuration file for LoToS
DemoMenuConfig.xml	Configuration file for Lotos (configures the Demo button).
demo1.xml	Configuration file for Lotos (topology associated with a Demo button).
Security files:	
testcert.cer	Certificate
.keystore	Keystore file that contains the private key contained in the certification file
Scripts:	
bin/lotos.sh, bin/lotos.bat	starts the Lotos application
bin/version.sh, bin/lotos.bat	displays the version of Lotos
Information:	
Readme-ZIP	Readme file in ASCII format
LotosUserManual	User manual in Word format (this file).

NOTE: If *hypercast.jar* or *hypercast-pastry.jar* is not in the ZIP archive, you need to copy them from the HyperCast ZIP archive or the HyperCast-Pastry ZIP archive, respectively.

NOTE: Lotos assumes that these files are in the directory where Lotos is started. If the files are not found, Lotos gets the files from the lotos.jar archive.

Configuration files

There is no need to edit the configuration files.

Starting a Simulation

Lotos assumes that the *.xml files, the *.cer file, and the .keystore files are in the directory where Lotos is started. If the files are not found, Lotos gets the files from the lotos.jar archive.

From the command prompt

a) The commands are different in Unix shells and the DOS shell.

In Unix:

```
   >export              CLASSPATH=$CLASSPATH:lib/lotos.jar:lib/hypercast.jar:
                  lib/hypercast-pastry.jar:lib/bcprov-jdk14-
122.jar:lib/xalan.jar
```

In DOS:

```
   >CLASSPATH=%CLASSPATH%;lib/lotos.jar;lib/hypercast.jar;
                  lib/hypercast-pastry.jar;lib/bcprov-jdk14-
122.jar;lib/xalan.jar
```

b) Display the version of Lotos with:

```
   >java lotos.Version
```

c) Run the simulator with the command:

```
  >java lotos.GUI
```

As Applet

Open the file /lib/lotos.html from a web browser. All jar files must be in the same directory as the lotos.html.

The Internet browser must use a JVM with Java version 1.4.x or higher, or the simulator will not load. To check the JVM version of your browser, open this URL with your browser http://javatester.org/version.html

Using Scripts

There is a set of Unix scripts (*.sh) and DOS scripts (*.bat) that can help with starting the above applications.

a) Set the directory /bin into your PATH environment variable:

In Unix:

```
>export PATH=$PATH:bin/
```

In DOS:

```
>PATH=%PATH%;bin/
```

b) Then run

```
>lotos.sh
```

```
>version.sh
```

Under Windows, you can go to the /bin directory and click on the *.bat files.