



Java¹ Simulator for S-TGDH² : proof of concept (POC) Presentation ³ of the simulator

July 20, 2010

Author : Julien Thomas

1 Programming Language developed by SUN, see <http://java.sun.com>

2 S-TGDH, secure enhanced group management protocol in ad-hoc networks

3 A full documentation, generated with doxygen, is also available at <http://master.julienthomas.eu>

Java Simulateur *for S-TGDH†: proof of concept (POC)

Julien Thomas

July 21, 2010

Contents

Contents	2
I Presentation	3
II Modeling and Programming the STGDH protocol	3
II.A Node Simulation	3
II.B Simulating the Network	4
II.C Remaining Work	6
III Using the simulator	7
III.A The different modes, launching the simulator	7
III.B the graphical mode	7
III.C The <i>command-line</i> mode	10
III.D Available languages	10
III.E Extending the simulator	11
IV Problems while using the simulator	11
IV.A Graphical (swing) error raised when starting nodes	11
IV.B Failure of the initialization	13
IV.C UI problem	13
IV.D Returning to the beginning of a line in console mode	13
References	13

*Programming Language developed by sun, see <http://java.sun.com>

†S-TGDH, secure enhanced group management protocol in ad hoc networks [FC07]

I Presentation

The S-TGDH project aims at simulating the S-TGDH protocol, as a proof of concept. S-TGDH is a secure version of TGDH, TreeGroup Diffie Hellman, a group management protocol for ad hoc networks. In our solution, we defined additional properties such as node authentication, in order to make the algorithm more robust. The mutual authentication relies on a simple shared password (or passphrase) defined and shared at the beginning of the construction of the group, which makes the protocol lightweight and thus adapted solution for to common usages.

S-TGDH also proposes optimized algorithms and solves a problem related to the management of concurrent operations due to the distributed dimension of the considered environment.

The goal of this simulator is to show that our solution may be implemented in real environment and thus is not a simple unrealistic concept. As this simulator is a proof of concept, it may not be optimized in term of algorithmic choices / or programming, though I tried to minimized such un-optimizations while programming it.

For more information, please

- read the article on S-TGDH for CRiSIS'07 : *S-TGDH, secure enhanced group management protocol in ad hoc networks*, Frederic Cuppens, Nora Cuppens, and Julien Thomas, The International Conference on Risks and Security of Internet and Systems, CRiSIS'2007, Marrakech-Morocco 2-5 July 2007
- contact me (julien [DOT] thomas [AT] telecom-bretagne.eu)
- see my website <http://www.julienthomas.eu/> and more precisely <http://master.julienthomas.eu/>

II Modeling and Programming the STGDH protocol

II.A Node Simulation

The different classes of the *node* package aim at managing the nodes and their behaviors, as described by the UML schema figure 1. The main class, *Node.java* is an extension of the JAVA class *Thread*, in order to launch nodes in independent threads. The other classes are behavior classes, defined to implement for the time being the initialization steps (*initNode* behavior) and the group management steps (*groupNode* behavior).

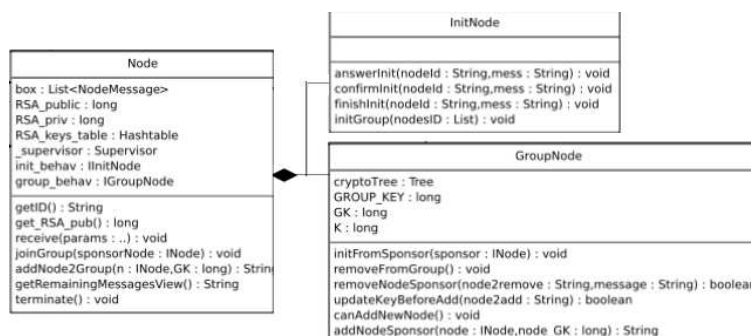


Figure 1: Modeling a node of the network

Node.java Class

Node.java is the main class of the node package and is responsible for the management of each node and the messages it receives. In order to easily manage and control the messages sent to each node,

the *Node* class has been built to receive any of the messages and then dispatch them to the relevant behavioral subclass. This generates a more complex node class and less object oriented but this simplifies the protocol analysis and the step by step development process I decided to rely on.

Behavior Classes

For the STGDH protocol, two classes have been designed to manage the different steps of the protocol. *InitNode.java* models the behavior of a node when considering the construction of a group (protocol in 4 steps). The class relies on the following principal methods:

1. `public void initGroup(List<String> nodesID)`: The initiator node sends a *INIT_GROUP* to the other members of the group
2. `public boolean answerInit(String nodeID, String mess)`: The node sends a message of type *ANSWER_INIT_GROUP* to the source (after receiving a *INIT_GROUP* message from it)
3. `public boolean cancelInitGroup(String nodeID, String reason, boolean checked)`: The initiator node cancels the group establishment
4. `synchronized boolean confirmInit(String nodeID, String mess)`: The initiator node validates the *nodeID* authentication (challenge 1). When the initiator node received from all the nodes of the group the message *ANSWER_INIT_GROUP*, it sends a *FINISH_INIT_GROUP*
5. `boolean finishInit(String nodeID, String mess)`: Nodes validate the source authentication (challenge 2) and then emit a *FINISH_INIT_GROUP*

GroupNode.java models a node which belongs to the security group (in the simulator, only one group is possible and thus a node may belongs to a group only). The class relies on the following principal methods:

1. `public void init()`: When the group is created, this method is supposed to be called by each member
2. `public synchronized void initFromSponsor(INode sponsor)` the node is added by the sponsor node *sponsor*
3. `public synchronized void doAddNodeMessage(String nodeSponsor, String message)`: the *addNode* message is sent by the sponsor node *nodeSponsor*
4. `public synchronized boolean doRemoveNodeMessage(String nodeSponsor, String message)`: the *removeNode* message is sent by the sponsor node *nodeSponsor*

Managing messages inside a node

As previously said, the class *Node.java* is used to centralize the management of a node. It thus receives all the messages and then dispatches them, providing:

- the state of the node (see *NodeConstants.java*) is correct according to the message header. For example, a node in the state *NOTHING* is not supposed to management group messages.
- the message is signed and well signed, when such a signature is required.

When errors occur, the message is either deleted (incorrect signature) or delayed (*DelayException*) in order to consider both asynchronous and unstable communications.

II.B Simulating the Network

General Principles

The network mainly relies on the class *Supervisor.java* of the package *manager* and thus more generally on the package *manager*. The network works as follows:

1. A thread manages the message sent by the nodes and stores them in a queue associated to each source.
2. Upon each wake-up, this thread randomly chooses a queue and send the first message it contains to the relevant nodes (more precisely, the relevant node queues)

The instability of the network, summarized in the figure 2, is simulated as follows:

- An independent network management thread
- A random choice of the message to send, using timer on the the network management thread
- RTT based messages to force the propagation of unsent messages.

The second aspect of the ad hoc networks, the lack of infrastructure, is more a problem at the routing layer of the OSI model. This issue is not consider in our model as we only consider issues at the application layers, the underlying layers being abstracted by the memory based communication process.



Figure 2: Modeling the Network

Network Hopheses

An essential hypothesis has been done when considering the messages from a same nodes: they are ordered and they will conserve this order (unless the message is delayed by the intended node).

Managing the messages in the Network

Message are sent using the method `public synchronized void send(INode srcNode, String nodeDestID,String header, String message, String sign)` which may be described as follows:

- `srcNode` is the initiator node
- `nodeDestID` is the intended node
- `message` is the payload
- `sign` is the signature given by the source. It may be null
- `header` is the header of the message (related to the protocol) and must be set using the constants of the class `SupervisorMessage.java` in the package `manager`.

The messages management process also relies on other methods:

1. `public void run()`: method called each time the network management thread wakes up
2. `synchronized void delayMessage(String nodeSrcID, INode dest, String header, String message, String sign)`: called by a destination when the message is not appropriate when considering its current state

3. `void broadcastGroup(INode srcNode, String header, String message, String sign)`: similar to `send()` but the destination is set to `null`
4. `checkBox()` randomly chooses a message to send
5. `decreaseRTT()` and `forceSend()`: decreases the RTT of the messages and sends the messages whose RTT is elapsed

II.C Remaining Work

Advanced Management of the node adding algorithm

For the time being, the adding process is not optimized: virtual nodes could be replaced by real nodes. This comes from the current algorithm, which follows the following global update scheme:

1. sponsor choice
2. sponsor validation
3. synchronization sponsor - new node
4. sent of the `addNode` message from the sponsor

Though the global algorithm has to be maintained, the current concept prevent the following optimization: the node to add takes the place of the parent node if it is virtual and the left node is the sponsor. This issue comes from the fact the decision depends on the sponsor (see also the figure 3): the sponsor node determines the subtree to update and then propagates the messages to the group members.

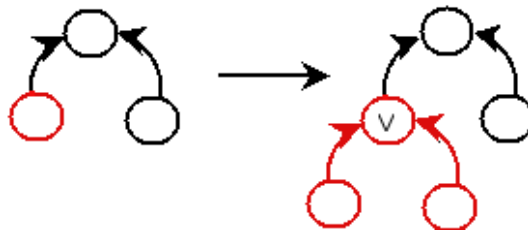


Figure 3: Current updated process: efficient when no virtual node

When considering a correct update, the adding process would produce the schema described in the figure 4. Were the right node performing adding/removal operations, conflicts would occur. The update messages must thus be reconsidered.

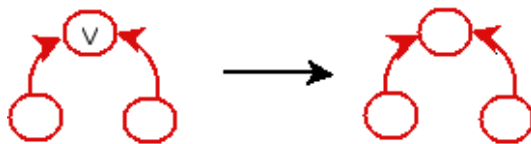


Figure 4: A possible problem when considering the optimized algorithm

Standard Ciphing Methods

Currently, ciphing techniques are simulated. This simulation was used as it prevented the cost of also implementing the ciphing algorithms, as it is not the main issues of the simulator. However, the usage of a ciphing libraries as `cryptlib` or `libcrypto` may be considered. A wide panel of the existing libraries is defined in a report of the Agence Nationale de Sécurité Informatique (ANSI)¹.

¹Sécurité des développements, Application Development Security [dSI06], section CryptoAPI

III Using the simulator

III.A The different modes, launching the simulator

The simulator may be run in two modes. The *graphical* mode lets users perform tests and evaluate on-the-fly the algorithm and its evolution: sent message, sponsor evaluations, ... The *command-line* mode lets users randomly evaluate the algorithms using randomly generated operations on random intervals of times. In this second mode, no interaction is possible after the simulation is launched. Only the initial configuration is managed by the user. Additional informations are also available using the command `java <program> -jar help`:

Simulator parameters:

```
-lL try to start the launcher in the L language
-mM try to start the launcher in the M mode. Supported mode are: 1 for console,
2 for graphical
```

Additional options

The console mode supports:

```
-dV the simulation duration in seconds
```

Examples

```
<<program>> -lfr -m2, to launch the french version of the simulator, in graphical mode
<<program>> -len -m1 -d500, to launch the english version of the simulator, in console
mode, for a duration of 500 seconds
```

The two mentioned modes are described in the next subsections.

III.B the graphical mode

The *graphical* mode is launched using the option `-m2`. It is characterized by a main window as illustrated by the figure 6. As described on the figure 5, several main components have been defined.



Figure 5: Several Component are available when using the *graphical* mode

1. *Actions principales / Main Actions*. The figure 6 illustrates the main panel of the simulator. Using it, it is possible to perform network management operations: add a node, create a security group, add/remove node from the security group.



Figure 6: Main panel, actions on nodes of the network

2. *Informations du groupe / Group Informations.* Several informations concerning the security group are available, as described in the figure 7. These informations let the user evaluate the group and its evolution.

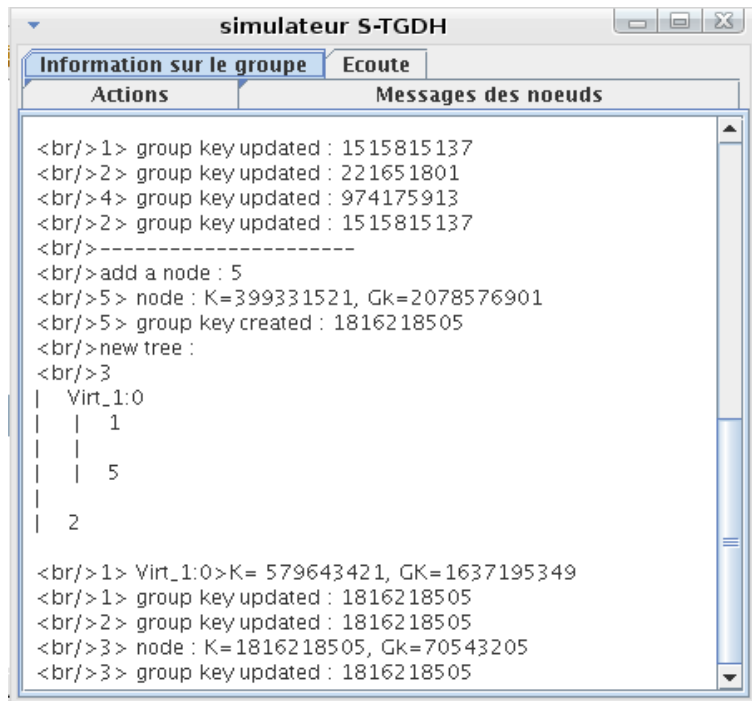


Figure 7: General messages from the nodes, illustrating the group state

3. *Messages des noeuds / Messages from the nodes.* It is also possible to obtain the information

associated to each node, using the component *Messages des nœuds* (figure 8). In order to obtain the information on a node, the user only needs to click on its ID.

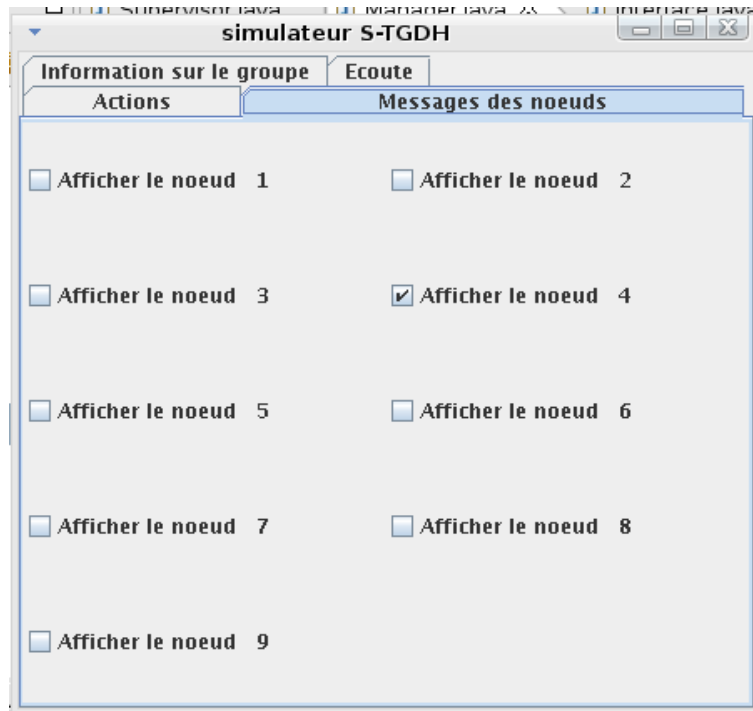


Figure 8: Nodes List, access to their informations

It is then possible to access the message it sends and receives, as illustrated by the figure 9. It is also possible to obtain the current state of the node using the *Actions* sub-panel.

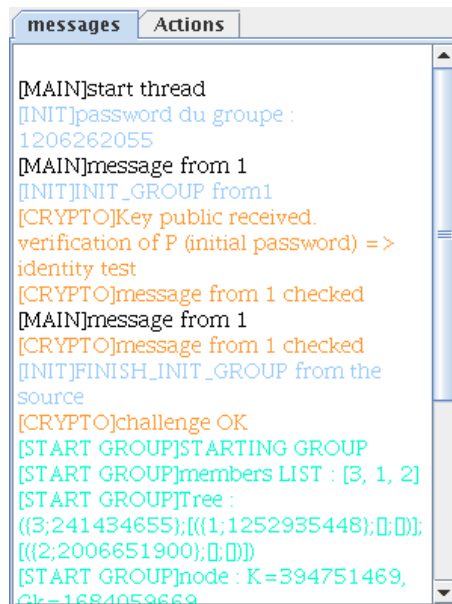
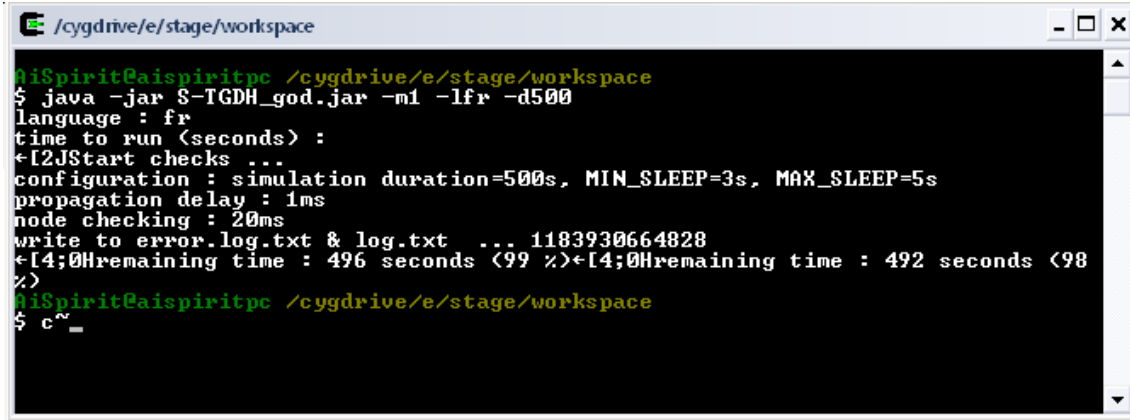


Figure 9: Informations of a node

III.C The *command-line* mode

The *command-line* mode (aka Console Mode) is launched using the option *-m1*. It has been developed to evaluate the stability of the algorithms, and for instance the asynchronous communications the simulator must manage. When run, it uses a random choice generator to perform both random actions (node entry, node entry in the group, node departure) on random intervals of times. The figure 10 illustrates the execution using the *command-line* mode of the simulator.



```
HiSpirit@aispiritpc /cygdrive/e/stage/workspace
$ java -jar S-TGDH_god.jar -m1 -lfr -d500
language : fr
time to run (seconds) :
←[2JStart checks ...
configuration : simulation duration=500s, MIN_SLEEP=3s, MAX_SLEEP=5s
propagation delay : 1ms
node checking : 20ms
write to error.log.txt & log.txt ... 1183930664828
←[4;0Hremaining time : 496 seconds (99 %)←[4;0Hremaining time : 492 seconds (98
%)
HiSpirit@aispiritpc /cygdrive/e/stage/workspace
$ c~_
```

Figure 10: Running the Simulator in the Console (Terminal) mode

This mode generates two log files:

1. *log.txt_N*, contains the messages sent by the nodes and is thus similar to the logging panel of the simulator when launch in the *graphical* mode
2. *error.log.txt_N* contains the list of the action performed by the console, as the error messages.

N is a random number generated to create unique files. It relies on the current time of the system (method `System.currentTimeMillis()` of JAVA). For the time being a rendering problem in the *command-line* tool may occur, depending on the operating system. This problem is explained in the subsection IV.D.

III.D Available languages

For the time being, the simulator is available in two languages: French and English (see section III.A which deals with the simulator configuration). The goal behind this internationalization of the simulator was to be able to produce reports in both French and English. Besides, this lets more people use the simulator.

In order to support this two languages, the internationalization technique has been used. Its principle may be describe by the following source code;

```
Locale locale = new Locale(language);
ResourceBundle res;
res = ResourceBundle.getBundle("STGDH", locale);
if(res == null) {
throw new LanguageException("undefined language " + language);
}
```

which make Java look for the file named inside the project *STGDH_language.properties* (i.e. the root of the project or the jar archive). The file relies on the following convention: *Leftrightarrow cle = value*. For instance

```
CREATE_A_GROUP = Creer un groupe
```

The management of the keys is defined in the package *lang*, class *Language.java* and relies on the following JAVA method:

```
(String)res.getObject("CREATE\_A\_GROUP")
```

Currently, the defined files are thus *STGDH_fr.properties* and *STGDH_en.properties*. However, it is possible to add new language by creating the associated *.properties* files (using existing ones). The usage of the new language defined by *X.properties* is done by launching the simulator with the *-LX* option, as presented in the section III.A.

III.E Extending the simulator

It is possible to use and extend the simulator, providing the license is respected (see the information associated to the source code). When considering the initialization, the simulator relies on the package *manager*, class *Launcher.java*, which in turn relies on the class *Launcher.java* of either the package *console* or *ui*, depending of the launch mode. It is thus possible to add additional mode and extend the class *Launcher.java* of the package *manager*.

When considering the communication process, it is possible to completely modify the current process in order to for instance rely on concrete protocol (TCP/IP for instance). As mentioned in the subsection II.B, the simulator does not consider the OSI layer below the application layer. However, as mentioned in the same subsection, this has been explicitly designed in the *Supervisor.java* class. In order to eventually implement routing protocol or protocols at other layers of the OSI model, it is possible to update the *Supervisor.java* class. For instance, to implement the OLSR protocol, it is possible to update the class as follows:

1. First, update `send(INode srcNode, String nodeDestID, String header, String message, String sign)` in order to add the underlying OLSR layer
2. Second, update `checkBox()` and `forceSend()` to either call an OLSR based routing method `propagateMessage(Message)` or the `doSendOperation(Message)` if the message is in the appropriate queue.

If one want to consider existing protocol implementations (OSLR or others), it is also possible ignore the *Supervisor.java* class and update the public `synchronized void receive(String header, String message, String signature, String nodeSrcID)` method of the *Node* class. However, the following informations from the *Supervisor.java* class will have to be reconsidered

```
private List<String> groupIDs; (initialized in initNode and used in Supervisor when  
    considering broadcast message)  
send() (used in both initNode and groupNode)  
broadcast() (used in both initNode and groupNode)
```

However, before doing so, one should wonder if the simulator is the best tool to simulate his protocol. Indeed, as previously mentioned, this is a proof of concept. Other tools as NS-2 may be more suited.

IV Problems while using the simulator

In this section are presented several problems encountered while using the simulator. Some of them were solved. If not, advice were given in order to prevent them from occurring.

IV.A Graphical (swing) error raised when starting nodes

```
language : fr  
Exception in thread "AWT-EventQueue-0" java.lang.ArrayIndexOutOfBoundsException: 0  
at javax.swing.text.BoxView.getOffset(BoxView.java:1077)  
at javax.swing.text.BoxView.childAllocation(BoxView.java:669)  
at javax.swing.text.CompositeView.getChildAllocation(CompositeView.java:215)  
at javax.swing.text.BoxView.getChildAllocation(BoxView.java:427)  
at javax.swing.plaf.basic.BasicTextUI$UpdateHandler.calculateViewPosition  
    (BasicTextUI.java:1879)  
at javax.swing.plaf.basic.BasicTextUI$UpdateHandler.layoutContainer
```

```

    (BasicTextUI.java:1855)
at java.awt.Container.layout(Container.java:1401)
at java.awt.Container.doLayout(Container.java:1390)
at java.awt.Container.validateTree(Container.java:1473)
at java.awt.Container.validateTree(Container.java:1480)
at java.awt.Container.validateTree(Container.java:1480)
at java.awt.Container.validateTree(Container.java:1480)
at java.awt.Container.validateTree(Container.java:1480)
at java.awt.Container.validate(Container.java:1448)
at javax.swing.plaf.basic.BasicTabbedPaneUI.ensureCurrentLayout
    (BasicTabbedPaneUI.java:1227)
at javax.swing.plaf.basic.BasicTabbedPaneUI.tabForCoordinate
    (BasicTabbedPaneUI.java:1268)
at javax.swing.plaf.basic.BasicTabbedPaneUI.tabForCoordinate
    (BasicTabbedPaneUI.java:1262)
at javax.swing.plaf.basic.BasicTabbedPaneUI$Handler.mousePressed
    (BasicTabbedPaneUI.java:3230)
at java.awt.Component.processMouseEvent(Component.java:5485)
at javax.swing.JComponent.processMouseEvent(JComponent.java:3126)
at java.awt.Component.processEvent(Component.java:5253)
at java.awt.Container.processEvent(Container.java:1966)
at java.awt.Component.dispatchEventImpl(Component.java:3955)
at java.awt.Container.dispatchEventImpl(Container.java:2024)
at java.awt.Component.dispatchEvent(Component.java:3803)
at java.awt.LightweightDispatcher.retargetMouseEvent(Container.java:4212)
at java.awt.LightweightDispatcher.processMouseEvent(Container.java:3889)
at java.awt.LightweightDispatcher.dispatchEvent(Container.java:3822)
at java.awt.Container.dispatchEventImpl(Container.java:2010)
at java.awt.Window.dispatchEventImpl(Window.java:1774)
at java.awt.Component.dispatchEvent(Component.java:3803)
at java.awt.EventQueue.dispatchEvent(EventQueue.java:463)
at java.awt.EventDispatchThread.pumpOneEventForHierarchy
    (EventDispatchThread.java:242)
at java.awt.EventDispatchThread.pumpEventsForHierarchy
    (EventDispatchThread.java:163)
at java.awt.EventDispatchThread.pumpEvents(EventDispatchThread.java:157)
at java.awt.EventDispatchThread.pumpEvents(EventDispatchThread.java:149)
at java.awt.EventDispatchThread.run(EventDispatchThread.java:110)

```

Possible Reasons

The most probable reason relies on a JAVA based problem of the thread synchronization, probably due to a too fast launch of the different threads when quickly performing a huge number of requests using the user interface commands.

Consequences

Some of the nodes fails during the initialization process (before the group creations). As no timer has been configured, the other nodes are waiting for the failing nodes ... indefinitely.

Solutions

Until the problem is solved (possibly by blocking the commands while a node is launching), wait for the user interfaces of the created nodes to be launched, between the creation of new nodes.

IV.B Failure of the initialization

While creating groups with 4 members, the last node of the tree does not initialize and freezes while calling the function *finishinit()* of *InitNode.java*, due to an unknown reason. It seems that the process is blocked by JAVA.

Consequences

Some of the nodes fails during the initialization process (before the group creations). As no timer has been configured, the other nodes are waiting for the failing nodes ... indefinitely.

Solution

The blocking came from a *synchronized* in the class *Supervisor.java*. IT has been removed and a better analysis of the synchronization dimension inside the simulator must be done. However, why did it block at 4 only? Mystery.

IV.C UI problem

When a node deletion fail, this is not expressed in the UI as it worked only in pushed mode (no control from the console on the user interface).

Solution

In order to avoid this issue, I added a communication interface in the package *ui: Interface.java*. It propose methods to partially control the user interface. The mentioned issues is thus solved by calling the relevant methods when a node deletion fails.

IV.D Returning to the beginning of a line in console mode

In the console mode, a return to the beginning of the line is performed to display simulation run percentages. Under linux/Unix, this is possible using the following special characters:

```
String coord = ESC+"4;0H";
System.out.print(coord+"remaining time : " + timeToRun +" seconds (" +percent+" %));
```

However, this only work with some command line tools. For instance, it is working under Linux but not under windows, event with Cygwin²).

Solution

For the moment, the only one solution consists in running the project in console mode using a POSIX compatible editor.

References

- [dSI06] Agence Nationale de Sécurité Informatique, *Sécurité des développements, application development security*, République tunisienne, Ministère des technologies et de la communication, August 2006, http://www.ansi.tn/fr/Guides_et_ressources/cours/Domaine8-SecDevloppment.pdf.
- [FC07] Julien Thomas Frederic Cuppens, Nora Cuppens, *S-tgdh, secure enhanced group management protocol in ad hoc networks*, The International Conference on Risks and Security of Internet and Systems, CRiSIS'2007, Marrakech-Morocco, LASS-CNRS, July 2007, pp. 83–92.

²Cygwin is windows based emulation of the Unix system. See <http://www.cygwin.com>