



Simulateur Java¹ pour S-TGDH² : preuve du concept

Rapport de présentation³ du simulateur

20 juillet 2010

Auteur : Julien Thomas

1 Langage de développement proposé par sun, voir <http://java.sun.com>

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

3 Une documentation de conception, générée à partir de doxygen, est également disponible sur le site de l'auteur, <http://aispirit.tuxfamily.org/~julienthomas>

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

Julien Thomas

20 juillet 2010

Table des matières

Table des matières	2
I Présentation	3
II Modélisation - développement	3
II.A Simulation d'un nœud du réseau	3
II.B Simulation du réseau	4
II.C Travail restant	5
III Fonctionnement du simulateur	6
III.A Les différents modes, lancement du simulateur	6
III.B Le mode graphique	7
III.C Le mode console	9
III.D Langages disponibles	10
IV Problèmes sur le simulateur	11
IV.A Erreur graphique de la JVM	11
IV.B Blocage de l'initialisation	12
IV.C Problème dans l'interface graphique	12
IV.D Problème de retour à la ligne	12
Références	13

*Langage de développement proposé par sun, voir <http://java.sun.com>

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

I Présentation

Le projet S-TGDH a pour but de simuler le protocole S-TGDH en tant que *proof of concept*. S-TGDH est une version sécurisée de TGDH, TreeGroup Diffie Hellman, un protocole de gestion de groupe développé pour les réseaux ad hoc. Dans notre solution, nous avons ajouté différentes propriétés comme l'authentification automatique des différents nœuds du groupe de sécurité, afin d'obtenir une version plus sécurisée. L'authentification mutuelle repose sur un simple mot de passe à configurer lors de la création du groupe, ce qui en fait une solution adaptée pour une utilisation courante et pour tous. S-TGDH fournit également des optimisations algorithmes et résout un problème de TGDH au niveau de la gestion d'opération simultanée en raison de l'aspect distribué de la solution : la mise à jour du groupe lors d'opérations a été modifiée.

Le but de ce simulateur est de montrer que notre solution peut être implémentée dans un environnement réel et n'est pas simplement un concept inadapté aux problématiques des réseaux. Comme il s'agit d'un *proof of concept*, le simulateur n'est pas forcément optimisé d'un point de vue algorithmique et différents choix peuvent ne pas être les meilleurs d'un point de vue optimisations algorithmiques / développement techniques, ce qui n'est pas le problème de notre simulateur (mais bon, ce n'est pas le but non plus).

Plus d'informations peuvent être obtenues

- par l'article sur S-TGDH pour la conférence 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
- en me contactant (julien [DOT] thomas [AT] telecom-bretagne.eu)
- en visitant mon site Internet, partie recherche : <http://www.julienthomas.eu/>

II Modélisation - développement

II.A Simulation d'un nœud du réseau

Les différentes classes du package *node* ont pour but de gérer le comportement des nœuds du réseau. Ceux-ci fonctionnent comme décrit par le schéma UML 1. La classe principale, *Node.java*, dérive de la classe java *Thread*, afin de lancer chaque nœud en tant que thread indépendant.

Classes principales

Node.java est la classe centralisatrice qui permet de contrôler les différents messages envoyés aux nœuds. L'approche est donc plus orientée modulaire avec liens entre les différents comportements que programmation orientée objet. Le but étant de pouvoir contrôler à partir d'une seule classe les messages arrivant sur un nœud. Cela m'a ainsi permis d'avoir un débogage simple, ainsi qu'une programmation itérative (ie au pas à pas, avec des tests intermédiaires), par ajout de fonctionnalités

InitNode.java modélise le comportement d'un nœud lors de la création du groupe (étape en 4 temps pour la synchronisation avec le sponsor et l'échange des clés).

GroupNode.java modélise un nœud appartenant au groupe de sécurité (dans la version actuelle, un seul groupe fonctionne sur le réseau et un nœud ne peut appartenir qu'à un seul groupe).

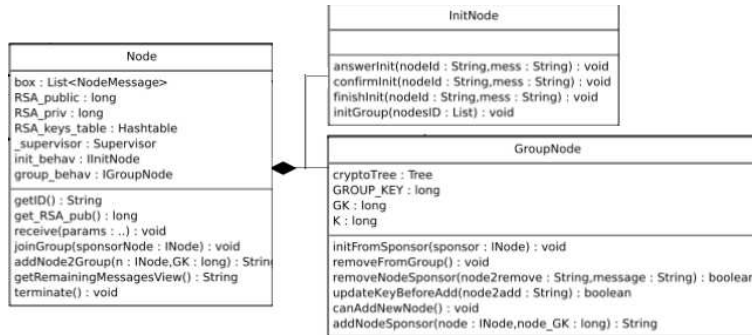


FIG. 1: Modélisation d'un nœud du réseau

Gestion des messages pour les différents modules

La classe `Node.java` permet de modéliser un nœud "normal" et de récolter les différents messages qui sont potentiellement retransmis aux modules "comportementaux", en fonction des conditions suivantes :

- l'état du nœud, (voir `NodeConstants.java`) est lié à l'entête du message. Par exemple un nœud dans l'état `NOTHING` n'est pas censé traiter des messages de groupes)
- le message est-il signé et bien signé, si cela est nécessaire (par exemple pour les messages lorsque le nœud est dans l'état `GROUP_MEMBER`)

Dans le cas où des erreurs surviennent, le message est soit rejeté (signature incorrecte, par exemple) ou mis en attente (`DelayException`) car il est possible que l'erreur soit due à un message qui n'a pas encore été traité.

Caractéristiques d'un nœud

Un nœud est donc caractérisé par

- Un thread indépendant
- Des messages asynchrones et indépendants (ordre partiel, seulement pour les messages d'une même source).

II.B Simulation du réseau

Principe général

Le réseau est principalement modélisé par la classe `manager/Supervisor.java` et plus généralement le package `manager`. Il fonctionne de la manière suivante :

Un thread supplémentaire récolte les différents messages émis par les nœuds (et stockés dans des files d'attente liées à chaque émetteur) du réseau et les traite à chaque réveil, en fonction de leur ordre d'arrivée ou non. En fait, seul l'ordre pour chaque message d'un même nœud est conservé, mais un choix aléatoire est fait entre les différentes files d'attente. Les messages sont alors envoyés aux différents nœuds, et plus exactement stockés dans leur file d'attente.

L'aspect instabilité des réseaux ad hoc est simulé par les caractéristiques suivantes, également résumés dans la figure 2 :

- Un thread indépendant,
- L'utilisation de timers pour représenter des délais de propagation (RTT),
- Des choix aléatoires entre les files d'attente pour représenter des délais instables et donc un réseau mobile.

Le deuxième aspect des réseaux ad hoc, l'absence d'infrastructure est plus un problème d'implémentation niveau 3 (routage, modèle OSI) que modélisation du réseau et donc intervient au niveau routage pour chaque réseau. Cet aspect n'a pas besoin d'être pris en compte dans notre simulateur car il est lié à d'autres problématiques tels que le développement d'un protocole de routage ou la gestion de la sécurité au niveau routage, et non à notre niveau, le niveau applicatif.



FIG. 2: Modélisation du réseau

Hypothèse du réseau

Une hypothèse essentielle est faite au niveau des messages d'un même nœud : Les messages émis par un nœud sont ordonnés et conservent toujours cet ordre (sauf si suppression par le destinataire). Si cela ne peut pas être assurée au niveau réseau, un estampillage des messages doit être effectué pour assurer un ordonnancement au niveau application. Des horloges de Lamport¹ peuvent être utilisées étant donné que seule une synchronisation au niveau de la source est nécessaire. Avec cet estampillage, la surcharge est minimisée et l'ordonnancement est assuré.

Catégories des messages du réseau

L'envoi d'un message sur le réseau s'effectue via la commande `public synchronized void send(INode srcNode, String nodeDestID, String header, String message, String sign)` qui peut être décrite de la manière suivante :

- srcNode est le nœud à l'origine du message,
- nodeDestID est le destinataire,
- message est le contenu utile (payload),
- sign est la signature apposée par l'émetteur. Elle peut être nulle,
- header définit l'entête du message. Les différentes entêtes sont définies dans le fichier `manager/SupervisorMessage.java`.

Note : `broadcastGroup()` contient des paramètres similaires et permet l'envoi de messages à tous les membres du groupe (ie. message multicast)

II.C Travail restant

Gestion avancée de l'ajout de nœuds

Pour le moment, l'ajout des nœuds est fonctionnel mais non optimisé : des nœuds virtuels sont créés alors qu'ils pourraient être remplacés par des nœuds réels. Cette situation est due à la conception actuelle de la mise à jour, comme détaillé plus loin. Une nouvelle version des ajouts est à prévoir.

La conception actuelle de l'ajout d'un nœud respecte le principal global de mise à jour suivant :

- 1° choix du sponsor
- 2° acceptation du sponsor
- 3° synchronisation sponsor - nouveau nœud
- 4° envoi du message aux membres du groupe

Bien que ce principe sera conservé dans les versions futures, la conception actuelle empêche l'optimisation suivante : le nœud à rajouter prend la place du parent s'il est virtuel, et si le nœud gauche est sponsor. Cela est dû au fait que la prise de décision doit être locale et non liée à d'autres nœuds réels. Or, pour le

¹voir l'article [Lam78]

moment, les messages actuels fonctionnent de la manière suivante (voir également la figure 3) : Le nœud sponsor détermine le sous arbre à modifier et le propage à tous les membres du groupes, afin de leur éviter d'effectuer un calcul quelconque.

De ce fait, l'ajout d'un nœud à la place du parent se traduirait par le schéma décrit dans la figure 4. Le nœud droite est donc associé à cette action et si celui-ci prend une action en même temps, des conflits surviendront.

Une remise en cause des messages, en respectant le principe de localité, permettront de résoudre ce problème propre à la conception parallèle, car les tests unitaires ne montraient pas ce problème.

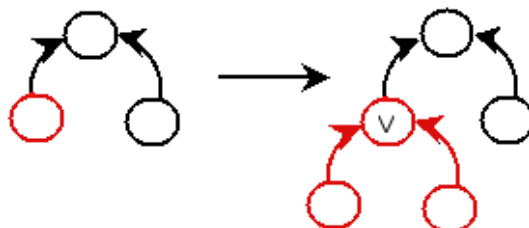


FIG. 3: Ajout correct d'un nœud dans le groupe

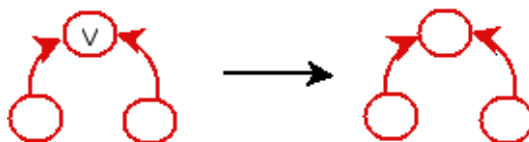


FIG. 4: Ajout d'un nœud : problème en exécution parallèle

Gestion normale des techniques de chiffrement

Actuellement, les techniques de chiffrements sont simulées et non réelles. Cela évitait d'avoir à gérer le chiffrement en plus de la programmation des algorithmes en eux-même. Toutefois, l'utilisation de bibliothèques de chiffrement comme *cryptlib* ou *libcrypto* peut être envisagée. Un large panel de ces bibliothèques est décrit dans le rapport de l'Agence Nationale de Sécurité Informatique (ANSI)². Des études comme la surcharge provoquée par les API et leur facilité d'utilisation (par exemple disponibilité en JAVA) seront essentielles.

III Fonctionnement du simulateur

III.A Les différents modes, lancement du simulateur

Le simulateur est disponible en deux modes. Le mode graphique permet de faire des tests pratiques sur le déroulement de l'algorithme : messages envoyés par chaque nœud, ajout d'un nœud en choisissant le sponsor ... Le mode console permet de tester les différents algorithmes en effectuant des opérations aléatoires à des intervalles irréguliers, pour tester la stabilité de l'algorithme et plus exactement du simulateur. Dans ce deuxième mode, aucune interaction de l'utilisateur n'est possible, seule la configuration initiale (par exemple la durée de la simulation) est gérée par l'utilisateur.

Plus d'informations sont également disponibles via la commande `java «programm» -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
```

²Sécurité des développements, Application Development Security [dSI06], section CryptoAPI

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

III.B Le mode graphique

Le mode graphique (-m2) est caractérisé par une fenêtre principale représentée sur la figure 6. Comme décrit sur la figure 5, différents composants principaux sont définis. Ils sont décrits dans les paragraphes suivants afin d'expliquer leurs intérêts et leur utilisation.



FIG. 5: Différents composants sont accessibles sur l'interface graphique

Actions principales

Le figure 6 représente la fenêtre principale du simulateur. A partir de cette fenêtre, il est possible d'effectuer les différentes opérations de gestion du réseau : ajouter un nœud sur le réseau, créer un groupe de sécurité, ajouter/supprimer un nœud du groupe de sécurité.

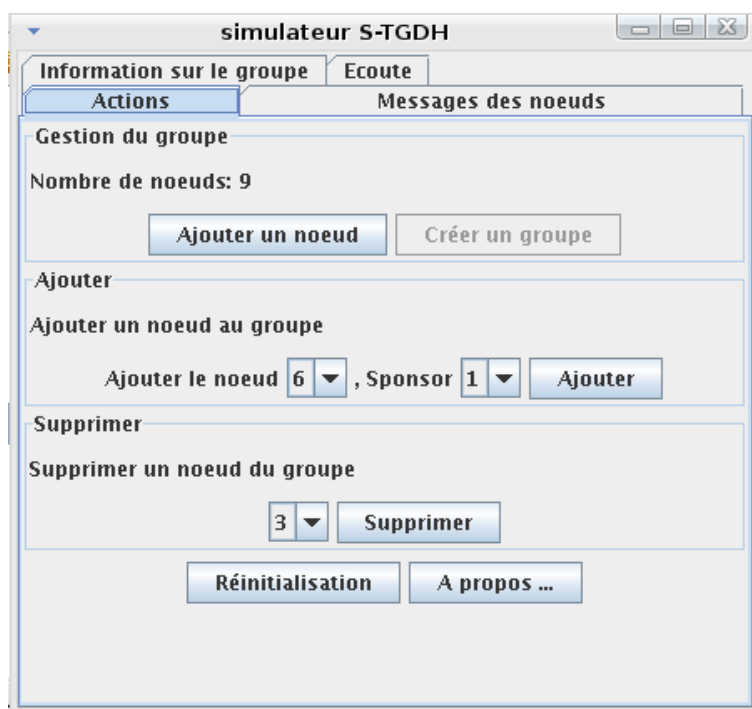


FIG. 6: Fenêtre principale, action sur les nœuds du réseau

Informations du groupe

Différentes informations générales sont fournis (figure 7) par les nœuds du groupe, plus précisément l'initiateur du groupe de sécurité ou les sponsors d'une opération. Ils permettent de voir l'évolution du

groupe et de vérifier que tout se déroule bien.

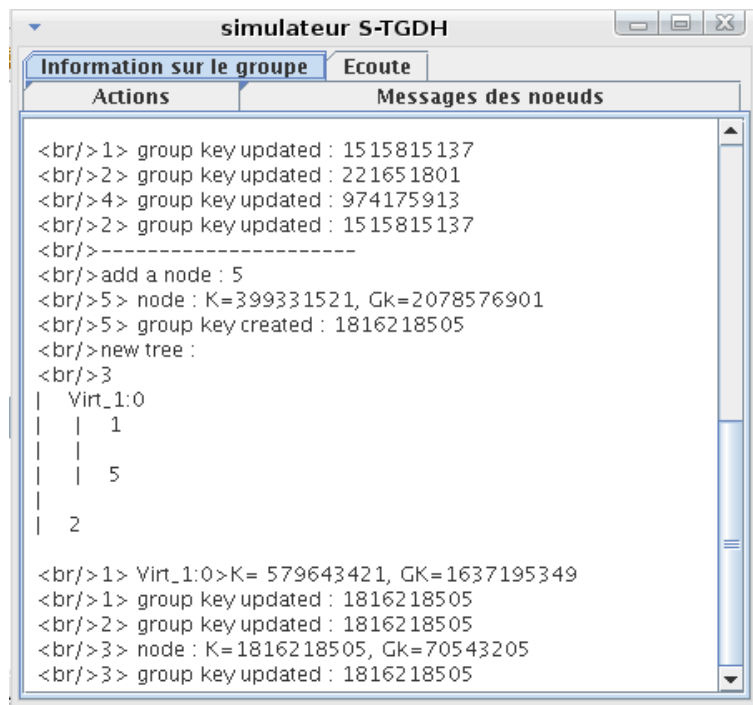


FIG. 7: Messages généraux des nœuds du groupe, représentant l'état du groupe

Messages des différents nœuds

Il est également possible de récupérer les informations propres à chaque nœuds, via l'interface *Messsages des noeuds* (figure 8). A partir de cette interface, il est possible de récupérer les informations d'un nœud en cliquant sur son identifiant.

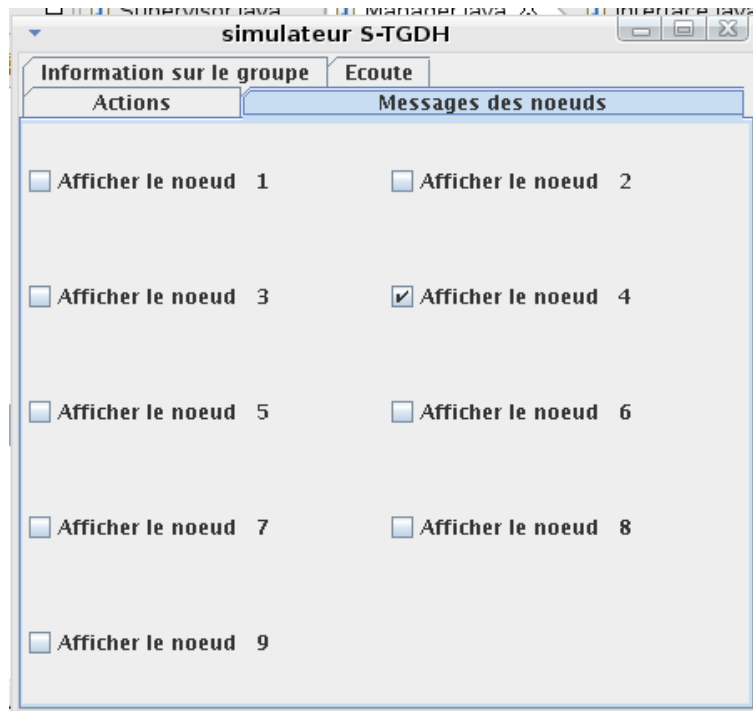


FIG. 8: Liste des nœuds du réseau, accès à leur informations

Il est alors possible de suivre les messages qu'il a reçu et qu'il a émit, comme l'illustre la figure 9. Il est également possible de récupérer l'état courant du nœud via l'onglet *Actions*

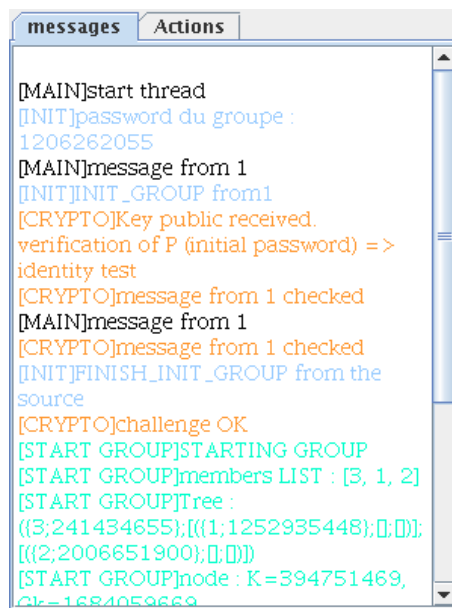


FIG. 9: Informations sur un nœud

III.C Le mode console

Le mode console (*-m1*) a été développé pour tester la stabilité de l'algorithme et notamment les opérations concourantes que doit résoudre le simulateur. Il permet de lancer de manière aléatoire, tant par la durée que par le choix des actions, différentes opérations sur les nœuds : entrée d'un nœud dans le

réseau, entrée d'un nœud dans le groupe et départ d'un nœud du groupe. La figure 10 illustre l'exécution en mode console du simulateur.

```

/cygdrive/e/stage/workspace
AiSpirit@aispiritpc /cygdrive/e/stage/workspace
$ java -jar $-TGDH_god.jar -m1 -lfr -d500
language : fr
time to run (seconds) :
Start 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
remaining time : 496 seconds (99 %)
remaining time : 492 seconds (98
%)
AiSpirit@aispiritpc /cygdrive/e/stage/workspace
$ c~_

```

FIG. 10: Exemple d'exécution en mode console

Ce mode génère deux fichiers de logs : l'un, *log.txt_N*, contient les messages envoyés par les différents nœuds, de manière similaire à l'interface graphique. L'autre (*error.log.txt_N*) contient la liste des actions effectuées par le mode console, avec d'autres informations annexes. N est un nombre aléatoire généré pour créer des fichiers uniques. Il se base sur l'heure système (*System.currentTimeMillis()* de Java).

Note : il existe un petit problème sur l'affichage dans le mode console, en fonction de l'OS utilisé. Ce problème est décrit dans le paragraphe IV.D.

III.D Langages disponibles

Le simulateur est disponible en deux langues : français et anglais (voir la section III.A sur la configuration au lancement). Le but de supporter ces deux langues et de pouvoir utiliser ce simulateur avec les rapports en français et donc une interface en français mais aussi en anglais avec donc l'interface en anglais cette fois ci. De plus, cela permet de disposer d'un simulateur utilisable par différentes personnes, sans avoir à utiliser l'anglais obligatoirement.

Pour supporter ces deux langues, la technique d'internationalisation de Java a été utilisée. Le fonctionnement peut être décrit par le bout de code suivant

```

Locale locale = new Locale(language);
ResourceBundle res;
res = ResourceBundle.getBundle("STGDH", locale);

if(res == null) {
throw new LanguageException("undefined language " + language);
}

```

qui fait que Java va rechercher dans le projet (ie la racine du projet ou de l'archive jar) le fichier nommé *STGDH_language.properties*.

Ce fichier respecte la convention suivante une ligne *Lefttrightarrow cle = value* avec par exemple

CREATE_A_GROUP = Créer un groupe

La récupération des différentes clés se fait alors via (*String*)*res.getObject("CREATE_A_GROUP")*.

Les différentes classes de gestion des langues sont disponibles dans le package *lang* et les fichiers de langue actuellement disponibles sont *STGDH_fr.properties* et *STGDH_en.properties*.

Il est à noter que d'autres langues peuvent être rajoutées au simulateur. Il suffit pour cela de prendre l'un des fichiers *.properties* existant et de changer les valeurs des *value*. La nouvelle langue est alors accessible en modifiant la valeur du paramètre *-m* (voir la section III.A) lors du lancement du simulateur

IV Problèmes sur le simulateur

Dans cette section sont listés les derniers problèmes rencontrés sur le simulateur. Ceux-ci sont soit partiellement corrigés (comme par exemple pour le problème IV.B) soit non corrigés mais dans ce cas les raisons sont évoquées et des conseils sont proposés pour éviter ces problèmes.

IV.A Erreur graphique de la JVM

```
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.EventQueue.pumpEvents(EventDispatchThread.java:157)
at java.awt.EventQueue.pumpEvents(EventDispatchThread.java:149)
at java.awt.EventQueue.run(EventDispatchThread.java:110)
```

Causes possibles

Un lancement trop rapide (clics trop rapides) des différents nœuds du réseau et donc threads. L'erreur étant interne à Java, il doit s'agir d'un problème de synchronisation / gestion des threads.

Conséquences

Certains des nœuds plantent à l'initialisation (avant création du groupe). Vu que les timers n'ont pas été configurés ... les autres ne le détectent pas et donc attendent en vain

Solutions

Attendre un peu que les interfaces graphiques se lancent, notamment pour la fenêtre principale du simulateur.

IV.B Blocage de l'initialisation

Lors de la création du groupe avec 4 membres, la dernière feuille de l'arbre n'effectue pas l'initialisation et bloque en plein milieu de la fonction *finishinit()* de *InitNode.java*, pour une raison inconnue. Il semblerait que le thread soit arrêté/bloqué par Java!

Conséquences

L'initialisation du groupe échoue car certains nœuds ne terminent pas l'étape 2 du processus et l'initiateur du groupe attend donc indéfiniment (pas de programmation par timer vu qu'il s'agit du PoC).

Correction du blocage

Le blocage était dû à un *synchronized* dans la classe *Supervisor.java* qui bloquait l'exécution. Je l'ai supprimé et une étude plus précise des différents *synchronized* du simulateur va être refaite. Le problème doit venir d'un blocage concourant avec l'interface graphique!

Pourquoi cela bloquait à 4 ... je ne sais pas. Probablement un comportement aléatoire, problème récurrent avec les programmations multi-threads...

IV.C Problème dans l'interface graphique

Lorsque la suppression d'un nœud échoue, cela n'est pas répercuté dans l'interface graphique et donc on pense que tout se passe bien alors que non.

Solution

Afin d'y remédier, j'ai rajouté une interface de communication dans le package *ui*, *Interface.java*, qui offre des points d'entrées dans l'UI. Ainsi, lors de la suppression d'un nœud du groupe, l'interface est averti et peut prendre

IV.D Problème de retour à la ligne

Dans le mode console, un retour à la ligne est effectué pour afficher un pourcentage d'exécution. Après quelques recherches, il semblait que la solution était d'ajouter un caractère spécial à l'affichage. Toutefois, ce caractère ne fonctionne qu'avec une partie des éditeurs de ligne de commande. Il est par exemple fonctionnel sous Linux, mais pas sous Windows (même avec Cygwin³).

³Cygwin est une collection de logiciels libres à l'origine développés par Cygnus Solutions permettant à différentes versions de Windows de Microsoft d'émuler un système Unix. (<http://fr.wikipedia.org>), voir <http://www.cygwin.com>

Solution

Pour le moment, la seule solution est d'exécuter le projet en mode console via un éditeur compatible POSIX, c'est à dire sous Linux/Unix ou Mac.

Évolutions

Actuellement, des modifications sont en cours pour pouvoir gérer le retour à la ligne peut importe l'éditeur de commande utilisé, ou du moins généraliser un peu plus cette fonctionnalité. De même, des informations supplémentaires style les opération effectuées vont probablement être rajoutées, si cela ne surcharge pas trop l'éditeur.

Références

- [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.
- [Lam78] Leslie Lamport, *Time, clocks, and the ordering of events in a distributed system*, Commun. ACM **21** (1978), no. 7, 558–565.