Simulation à événements discrets

Jonathan Lejeune

Sorbonne Université/LIP6-INRIA

ARA – 2018/2019

Inspiré des supports de cours de Sébastien Monnet

Systèmes répartis

Qu'est-ce ?

- Plusieurs unités de calculs distantes et autonomes
  - fiables/non fiables
  - homogènes/hétérogènes
- Un réseau les reliant
  - fiable/non fiable
  - communications synchrones/asynchrones

Pour faire quoi ?

- Stockage de données géo-répliquées
- Calcul parallèle
- Partage de ressources
- Répartition de charge
- Tolérance aux pannes

$\Rightarrow$ **Passage à l'échelle**

Systèmes répartis

Ce qui est transparent pour l'utilisateur :

- Localisation des ressources
- Accès aux ressources
- Hétérogénéité des ressources
- Pannes des ressources
- Extensibilité des ressources

Problèmes fondamentaux

- Diffusion fiable
- Checkpointing
- Réplication
- Consensus
- Exclusion mutuelle
- Élection de leader

} **Besoin d'algorithmes répartis**

Comment tester les algorithmes/systèmes répartis ?

Problèmes de tests sur des conditions réelles

- Nombre de sites généralement limités
  - $\Rightarrow$ Coût élevé
  - $\Rightarrow$ Réservation de ressources parmi plusieurs utilisateurs
  - $\Rightarrow$ Difficulté de tester le passage à l'échelle
- Indéterminisme
  - $\Rightarrow$ Résultats/bugs non reproductibles
- Fautes, latences
  - $\Rightarrow$ Environnement difficilement contrôlable
- Vision globale impossible
  - $\Rightarrow$ Monitoring difficile

Comment tester les algorithmes/systèmes répartis ?

Solution

Concevoir des plates-formes d'évaluation pour :

- Émuler/simuler un grand nombre de nœuds
- Injecter des fautes pour observer la réaction du système
- Vérifier des propriétés algorithmiques
- Évaluer le comportement/les performances du système

Simulation vs. Émulation

Émulation

Reproduction exacte par un logiciel du comportement d'un modèle où toutes les variables sont connues.

Simulation

Imitation d'un comportement physique réel et complexe en suivant un modèle abstrait qui extrapole des variables inconnues.

Peux-t-on simuler ou émuler

- un téléphone ? : émulation et/ou simulation
- un cyclone ? : simulation
- une chute d'un corps ? : simulation
- une machine ? : émulation et/ou simulation

Un comportement émulable est simulable. La réciproque est fausse

Plates-formes d'évaluation réparties

Caractéristiques

- Un ensemble de sites géographiquement éloignés et reliés par un réseau
- Chaque machine peut émuler plusieurs nœuds virtuels (selon sa capacité)
- Au sein d'une même machine, les communications peuvent être ralenties pour émuler le réseau

$\Rightarrow$ Possibilité d'avoir très grand nombre de nœuds (virtuels)

Un exemple de plate-forme répartie : PlanetLab

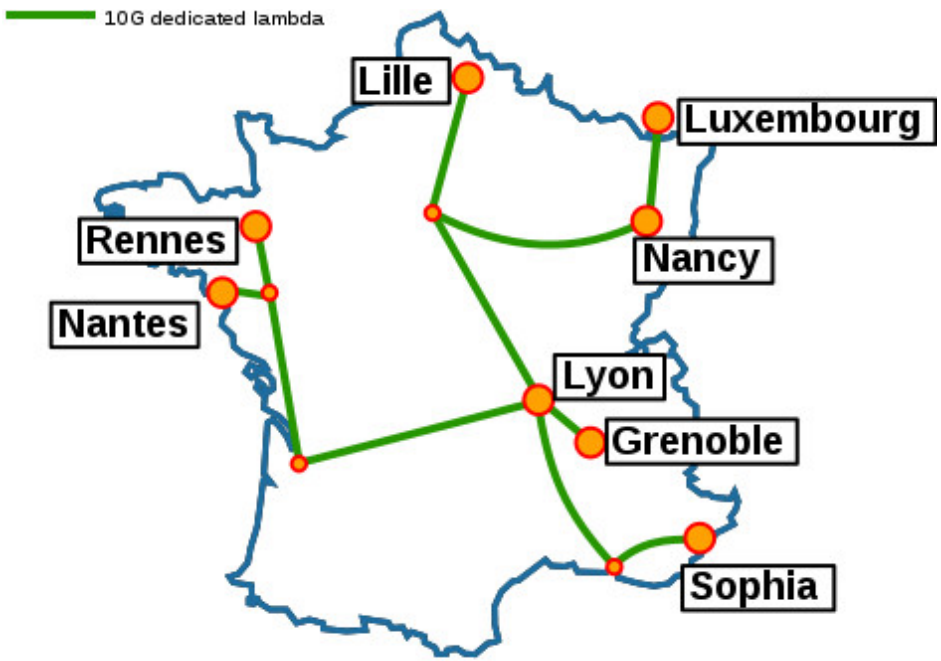
PLANETLAB
An open platform for developing, deploying, and accessing planetary-scale services

Caractéristiques à l'heure actuelle (2016)

- 1353 machines connectées par Internet
- 717 sites répartis sur toute la planète

Un autre exemple : Grid'5000

Grid'5000



Caractéristiques

- Plate-forme française
- 800 machines ($\simeq$ 12000 cores) réparties sur 8 sites
- Réseau par fibre optique, gigabit, infiniband

Avantages/Inconvénients des plates-formes réparties

Avantages

- Bonne montée en charge
- On peut faire tourner la vraie application sur les nœuds (virtuels)
- L'appli est réellement distribuée entre des sites physiquement distants

$\Rightarrow$ Proche d'un essai grandeur nature

Inconvénients

- Ressources précieuses
  - Il faut avoir accès à de telles plateformes
  - Il faut réserver les nœuds par avance et pour un temps limité
- Difficile à prendre en main
  - Des systèmes de déploiement complexes
  - Développement/débugage difficile !

Plates-formes centralisées : les simulateurs

Idée

- Mettre au point un modèle simplifié du système original
- Le simulateur s'exécute (en général) sur une seule machine qui simule l'ensemble des nœuds.
- Simplifications possibles : Application, OS, couches transport/réseau

$\Rightarrow$ mémoire d'une machine est généralement suffisante pour simuler des (centaines de) milliers de nœuds simplifiés.

Plates-formes centralisées : les simulateurs

Objectif

Tester **l'interaction** entre les nœuds du système :

- Les nœuds de l'application sont considérés comme des modules qui échangent des messages
- Il n'est pas toujours nécessaire :
  - de simuler le fonctionnement interne de chaque nœud
  - de représenter chaque donnée

Network Simulator (ns-{1,2,3})

Avantages

- Open Source (en C++)
- Simule précisément les protocoles réseau (TCP, WiFi, etc.)
- Très répandu dans la communauté réseau

Inconvénients

- Passage à l'échelle ?
- Très orienté "réseau"

OMNeT++

Avantages :

- Très générique : simule des modules qui échangent des messages
  $\Rightarrow$ Peut simuler des réseaux, des architectures multi-processeurs, des applications multi-threadées, etc.
- Permet de générer :
  - des graphes de séquence
  - une représentation graphique de la topologie
  - ...

Inconvénient :

Trop générique ?

SimGrid

Avantages :

- Très performant
- Modélisation du réseau réaliste
- Interfaces MSG, pseudo-posix et MPI
- Fonctionne sur le modèle d'un OS
- Écrit en C, et propose de nombreux bindings (Java, Ruby, ...)

Inconvénients :

- Prise en main difficile ...

Peersim

Avantages :

- Moins de 20000 lignes de code Java (Javadoc comprise)
- API très simple d'utilisation (comparé aux autres simulateurs)
- Peut se résumer à un moteur de simulation a événements discrets

Inconvénients :

- Parfois un peu trop simpliste (pas de simulation fine des protocoles réseau, ni même de gestion de bande passante)
- Gestion de topologie dynamique non intégré

En résumé :

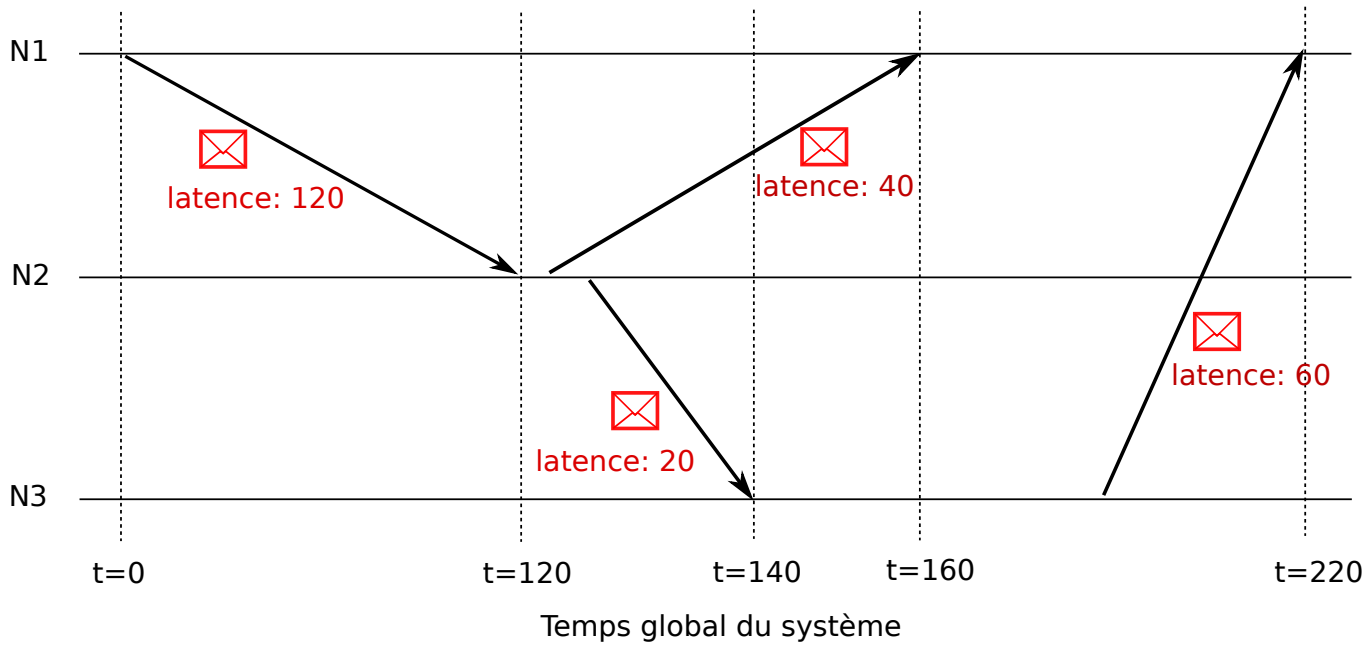
- Soit on teste le vrai système sur une plate-forme répartie
- Soit on conçoit un modèle simplifié et on teste dans un simulateur

La deuxième solution peut être satisfaisante dans de très nombreux cas.

Simulation à événements discrets

Idée : Discrétiser le temps

- Deux entités : les nœuds et les événements (e.g. messages)
- On considère que le temps évolue seulement lorsqu'un événement survient sur un nœud



Beaucoup de plates-formes de simulation sont basées sur ce modèle.

Gestion des événements

Principes

- Chaque événement/message :
  - est généré avec une estampille temporelle : **sa date de réception**
  - est inséré dans une file triée par estampille croissante
- Tant que la file est non vide ou temps max non atteint :

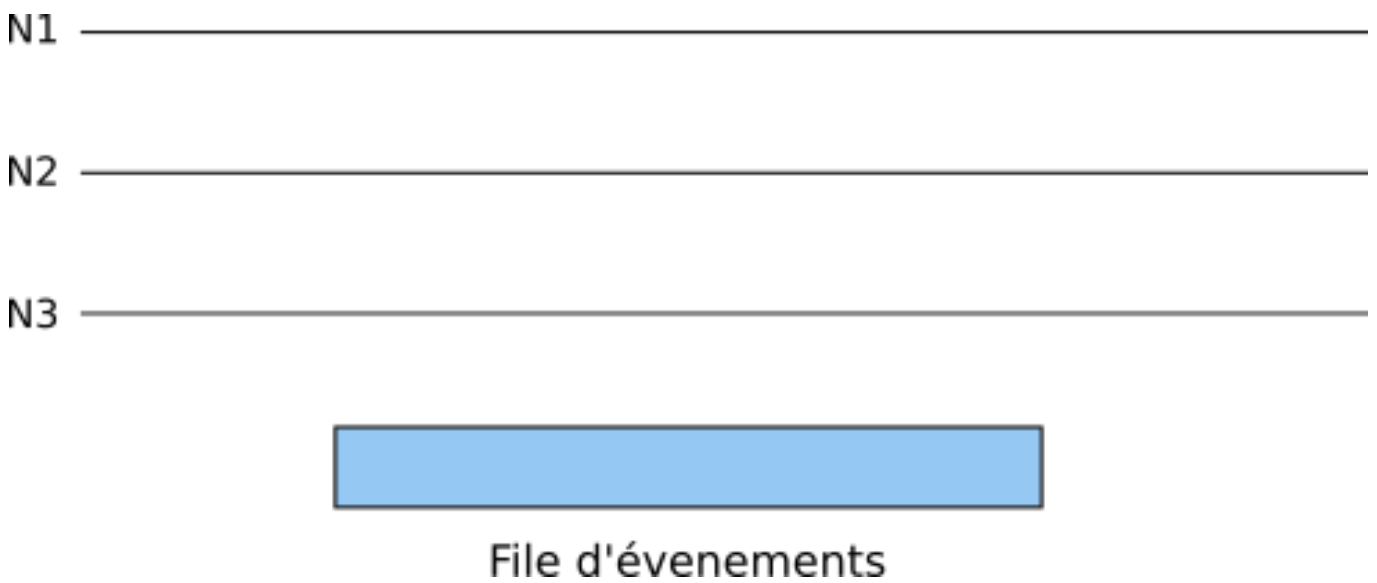
  récupérer l'événement *ev* en tête de file
  temps courant $\leftarrow$ estampille de *ev*
  délivrer *ev* au nœud destinataire

Gestion des événements

Principes

- Chaque événement/message :
  - est généré avec une estampille temporelle : **sa date de réception**
  - est inséré dans une file triée par estampille croissante
- Tant que la file est non vide ou temps max non atteint :

  récupérer l'événement *ev* en tête de file
  temps courant $\leftarrow$ estampille de *ev*
  délivrer *ev* au nœud destinataire

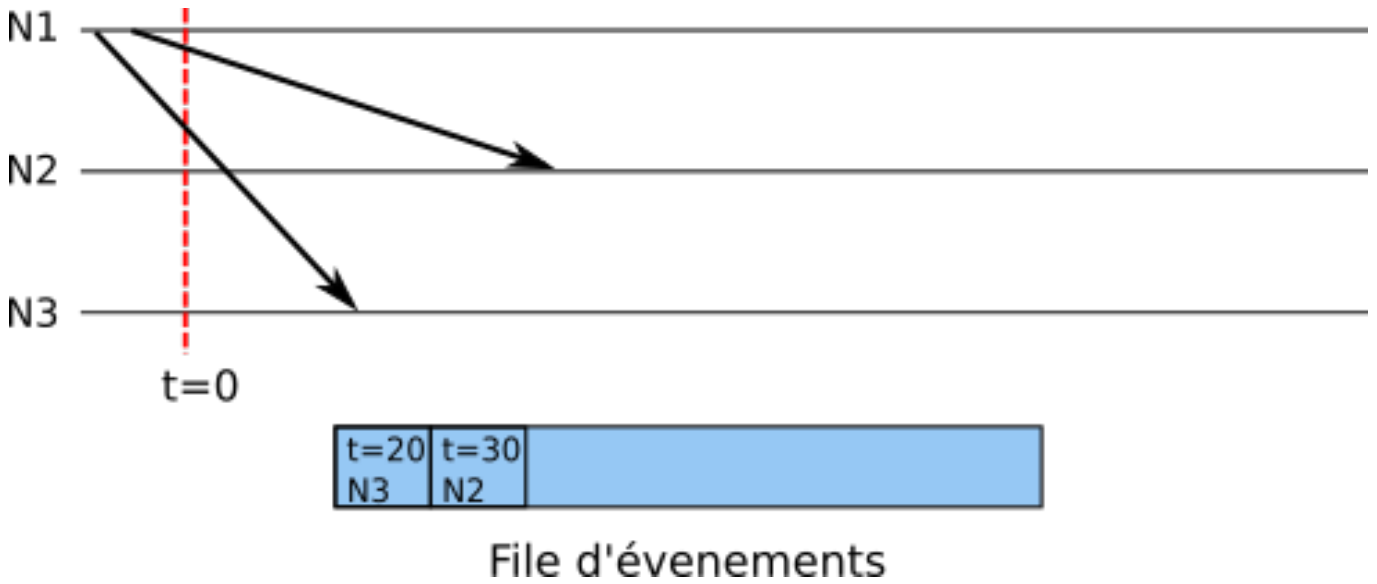


Gestion des événements

Principes

- Chaque événement/message :
  - est généré avec une estampille temporelle : **sa date de réception**
  - est inséré dans une file triée par estampille croissante
- Tant que la file est non vide ou temps max non atteint :

  récupérer l'événement *ev* en tête de file
  temps courant ← estampille de *ev*
  délivrer *ev* au nœud destinataire

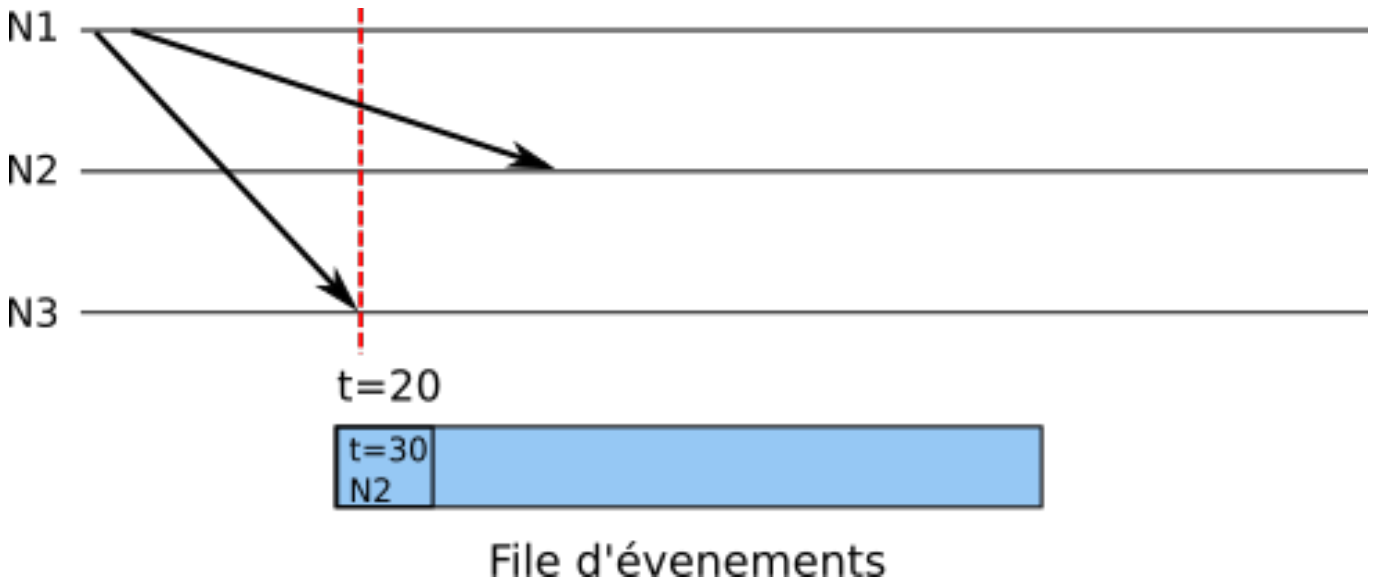


Gestion des événements

Principes

- Chaque événement/message :
  - est généré avec une estampille temporelle : **sa date de réception**
  - est inséré dans une file triée par estampille croissante
- Tant que la file est non vide ou temps max non atteint :

  récupérer l'événement *ev* en tête de file
  temps courant ← estampille de *ev*
  délivrer *ev* au nœud destinataire

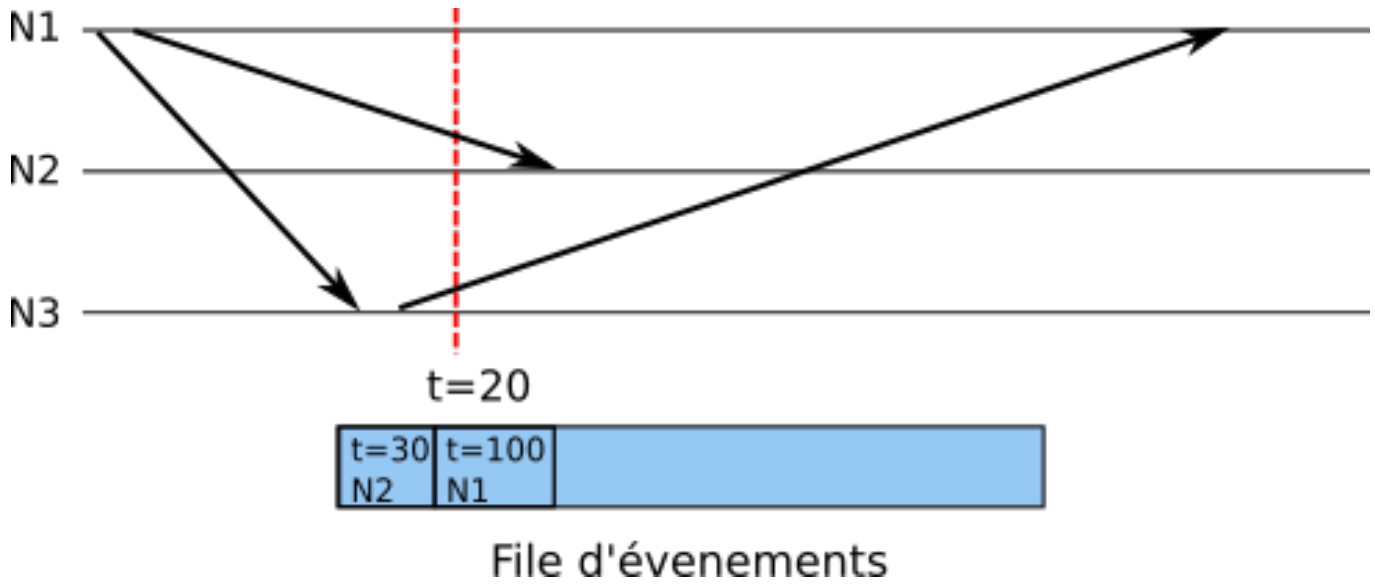


Gestion des événements

Principes

- Chaque événement/message :
  - est généré avec une estampille temporelle : **sa date de réception**
  - est inséré dans une file triée par estampille croissante
- Tant que la file est non vide ou temps max non atteint :

  récupérer l'événement *ev* en tête de file
  temps courant ← estampille de *ev*
  délivrer *ev* au nœud destinataire

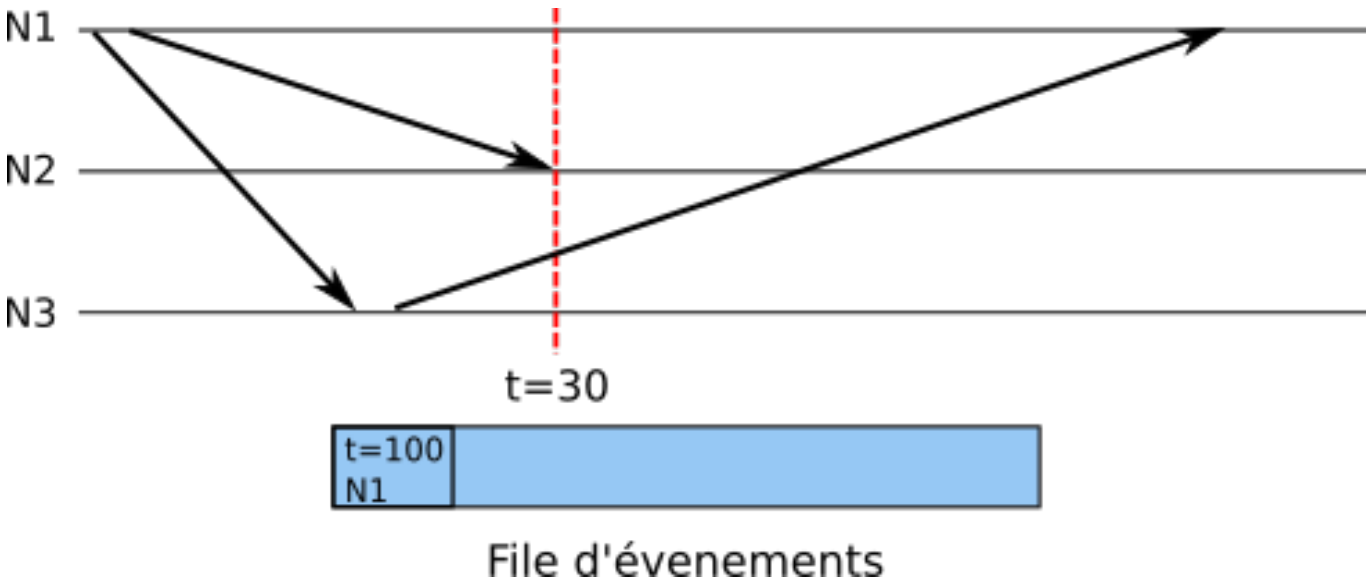


Gestion des événements

Principes

- Chaque événement/message :
  - est généré avec une estampille temporelle : **sa date de réception**
  - est inséré dans une file triée par estampille croissante
- Tant que la file est non vide ou temps max non atteint :

  récupérer l'événement *ev* en tête de file
  temps courant ← estampille de *ev*
  délivrer *ev* au nœud destinataire

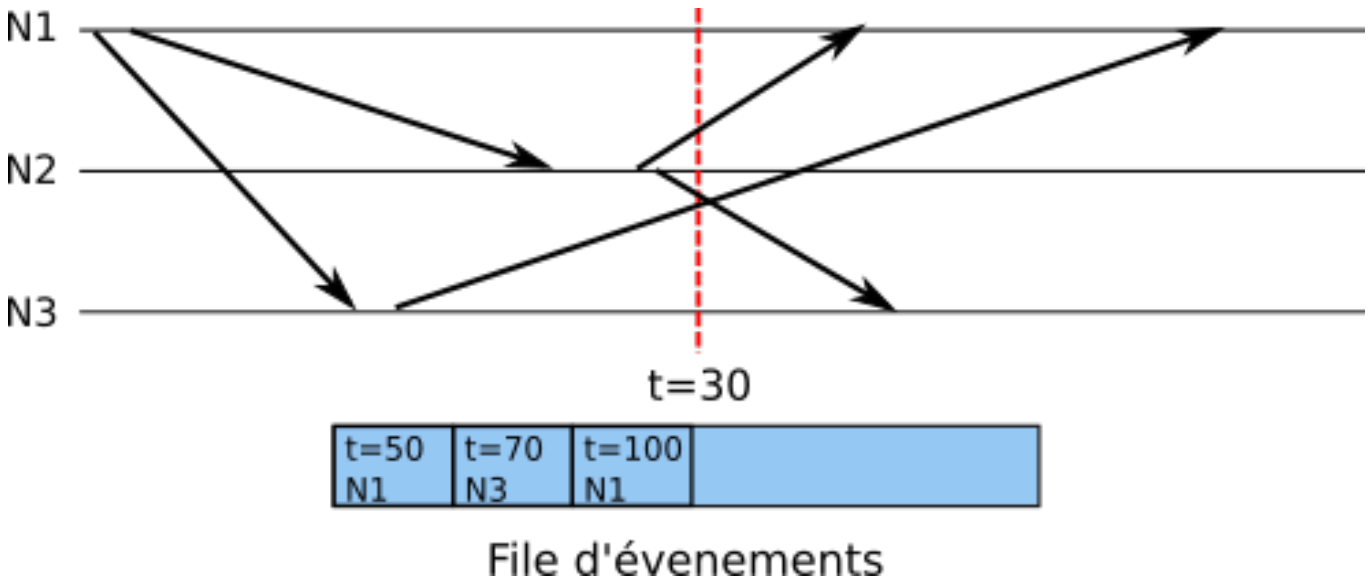


Gestion des événements

Principes

- Chaque événement/message :
  - est généré avec une estampille temporelle : **sa date de réception**
  - est inséré dans une file triée par estampille croissante
- Tant que la file est non vide ou temps max non atteint :

  récupérer l'événement *ev* en tête de file
  temps courant ← estampille de *ev*
  délivrer *ev* au nœud destinataire

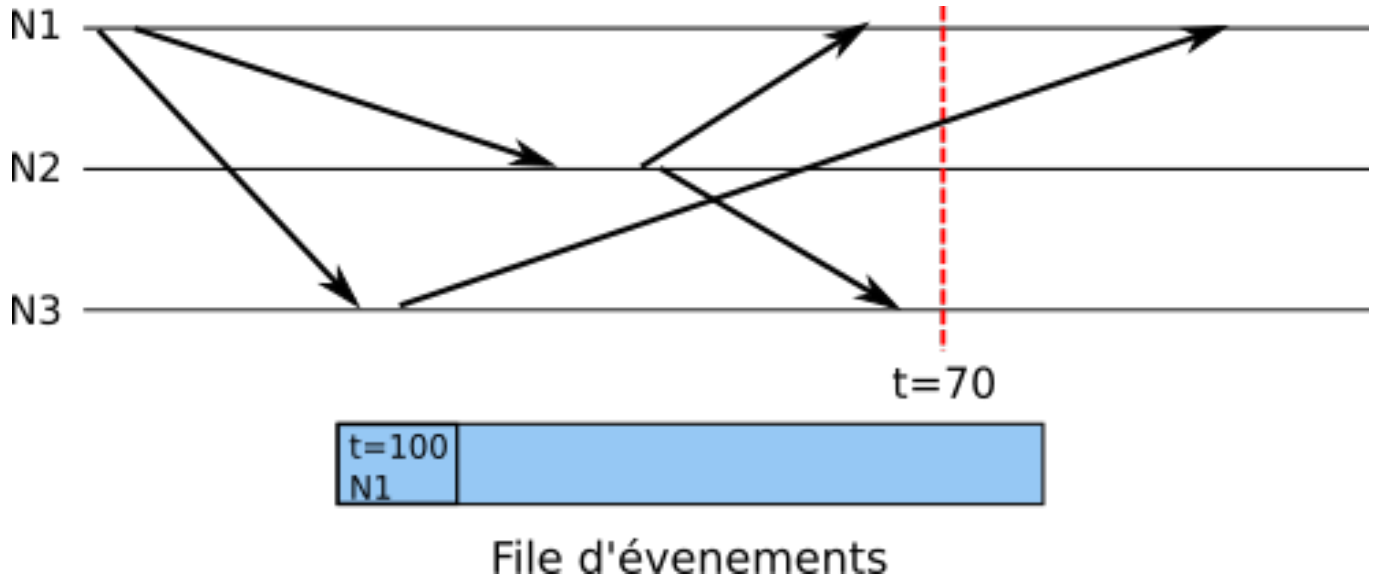


Gestion des événements

Principes

- Chaque événement/message :
  - est généré avec une estampille temporelle : **sa date de réception**
  - est inséré dans une file triée par estampille croissante
- Tant que la file est non vide ou temps max non atteint :

  récupérer l'événement *ev* en tête de file
  temps courant ← estampille de *ev*
  délivrer *ev* au nœud destinataire

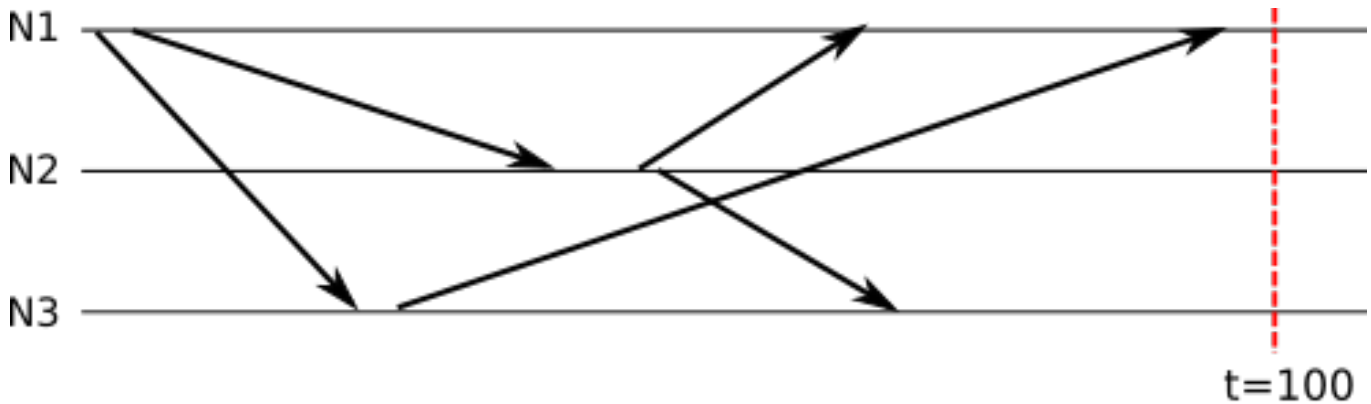


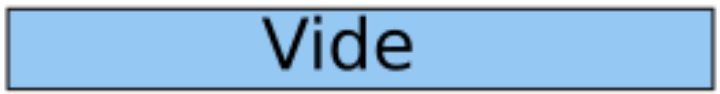
Gestion des événements

Principes

- Chaque événement/message :
  - est généré avec une estampille temporelle : **sa date de réception**
  - est inséré dans une file triée par estampille croissante
- Tant que la file est non vide ou temps max non atteint :

  récupérer l'événement *ev* en tête de file
  temps courant ← estampille de *ev*
  délivrer *ev* au nœud destinataire





File d'évenements

Gestion des événements

On simule en fait la réaction du système aux événements :

- Récéption de messages
- Événements internes aux nœuds

Mais pas le comportement du système entre les événements.

Avantages/Inconvénients

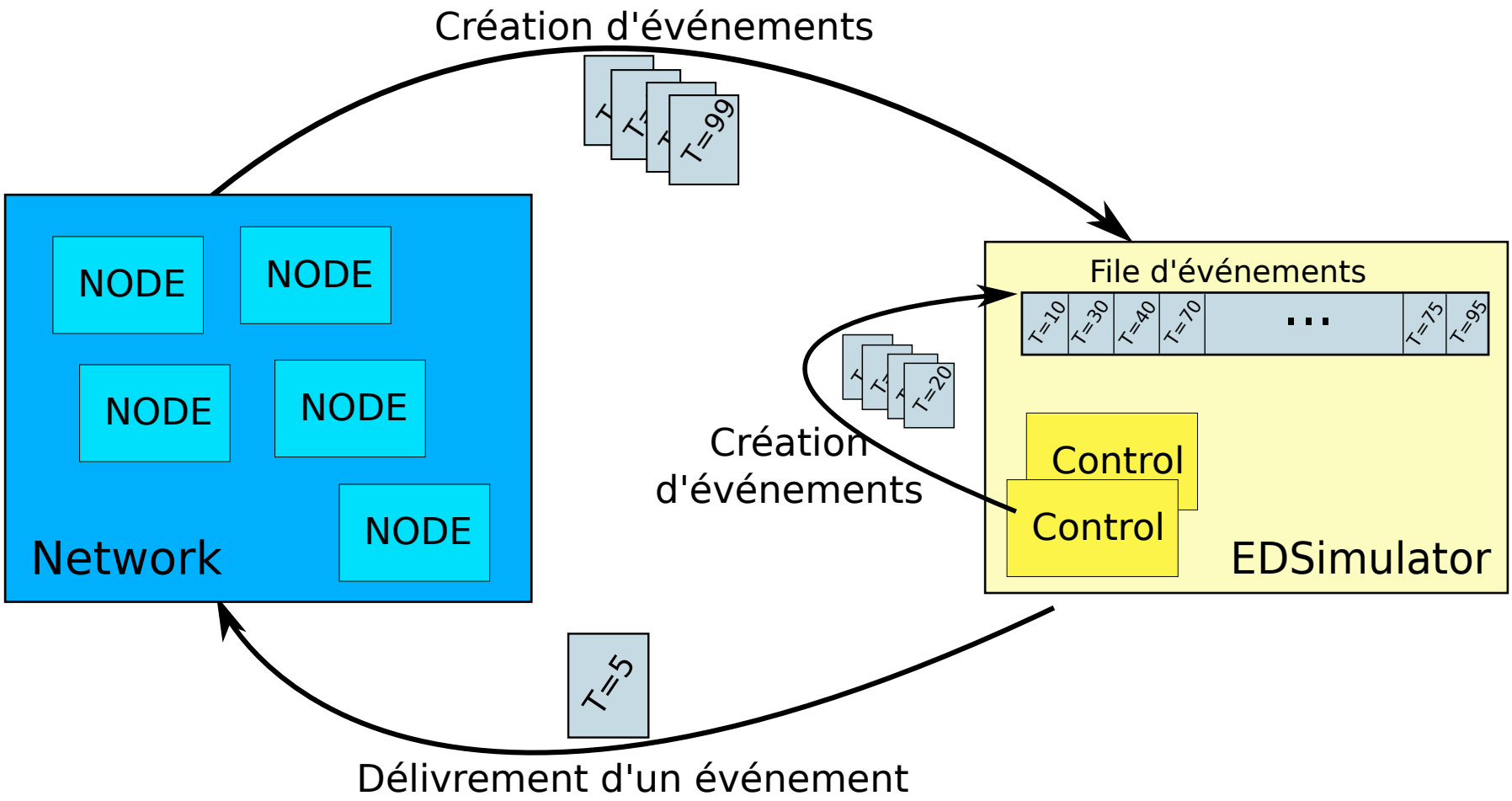
Avantages

- Le comportement du système entre les événements n'est pas simulé
  $\Rightarrow$ charge de calcul allégée
- Simulation reproductible
  $\Rightarrow$ le même bug peut être rejouer jusqu'à sa résolution
- Vision globale du système (message en transit + date de délivrance)
  $\Rightarrow$ debug facilité
- Tout se fait dans la même machine
  $\Rightarrow$ Possibilité de tricher

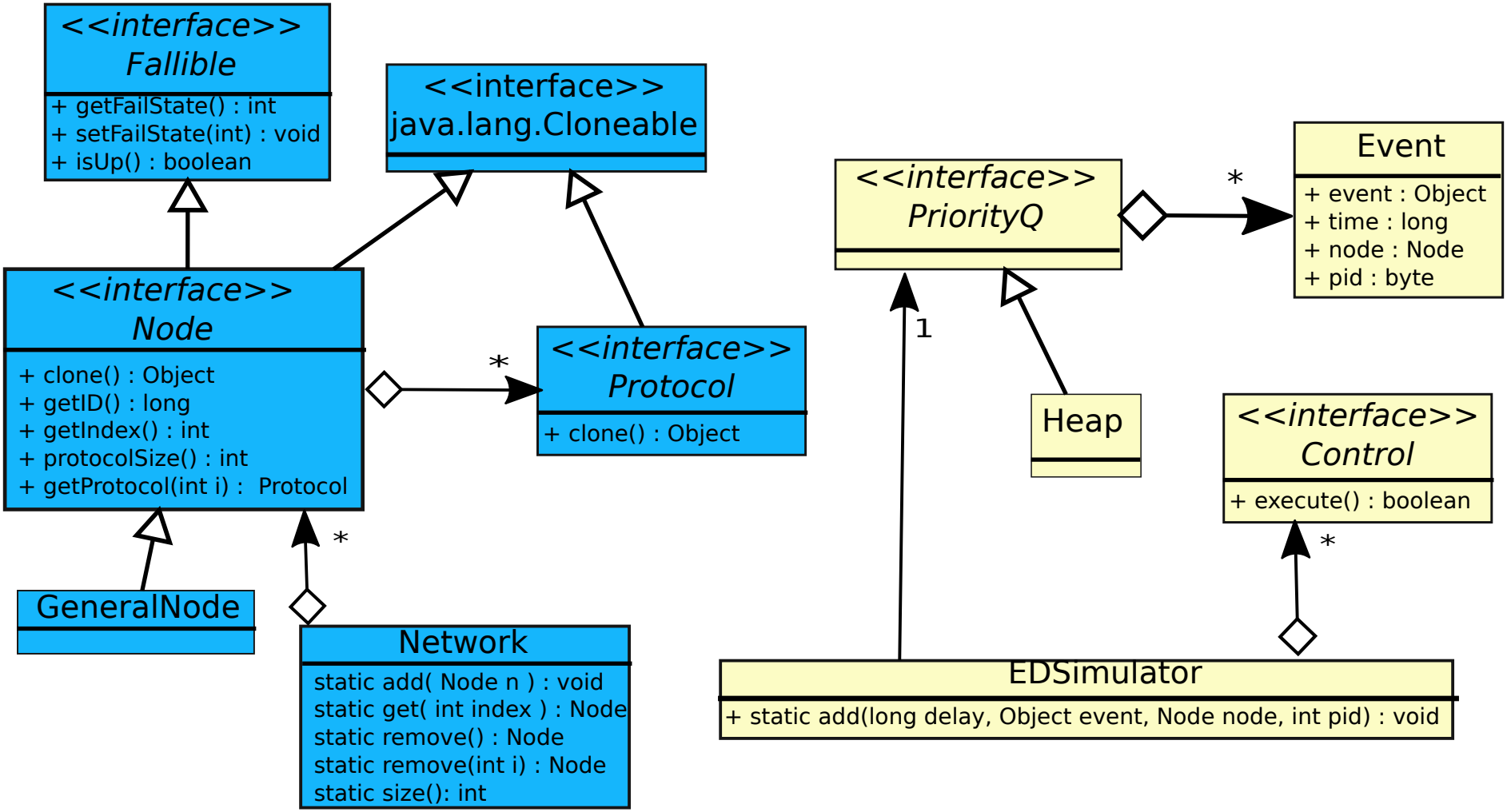
Inconvénients

- Trouver le bon compromis de finesse entre :
  - simulation précise au risque d'être ralentie par trop d'événements
  - simulation simpliste au risque de fausser le résultat

Vue d'ensemble du simulateur Peersim



Architecture principales du simulateur Peersim



Classe Network

Caractéristiques

- Classe singleton (que du `static`)
- Contient un tableau représentant l'ensemble des nœuds du système

Méthodes utiles :

- `Network.add(Node n)` : ajoute un nœud
- `Network.get(int i)` : retourne le nœud stocké à l'index i
- `Network.remove(int index)` : retourne et supprime le nœud à l'index i

Attention !

L'index d'un nœud dans le tableau ne fait pas office d'identifiant unique :

$\Rightarrow$ Peut être problématique en cas de système dynamique

$\Rightarrow$ Privilégier l'utilisation de l'ID du nœud

Interface Node

Caractéristiques

- Type représentant un nœud du système
- Autant d'instance que de nœud à simuler
- Les instances sont clonées (en profondeur) à partir d'un nœud prototype qui ne sert pas dans la simulation
- Classe concrète par défaut : `peersim.core.GeneralNode`
- Contient une collection de protocoles
- États possibles :
  - `Fallible.OK` : nœud opérationnel et accessible
  - `Fallible.DEAD` : nœud inaccessible définitivement
  - `Fallible.DOWN` : nœud inaccessible temporairement

Interface Node

Méthodes utiles :

- `setFailState(int codeEtat)` : change l'état du nœud
- `isUp()` : savoir si un nœud est opérationnel
- `getID()` : retourne l'identifiant unique du nœud
- `getIndex()` : retourne l'index actuel dans le tableau de `Network`
- `getProtocol(int i)` : retourne le protocole de type `Protocol` à l'indice i

NB : l'index d'un protocole au sein d'un nœud est le même sur tous les nœuds et fait office d'identifiant de protocole.

Interface `Protocol`

Caractéristiques

- Symbolise une couche protocolaire du système
- Une classe concrète de cette interface contient les routines du protocole
- Chaque instance de nœud a sa propre d'instance du protocole
- Chaque instance de protocol est créé par clonage à partir du nœud prototype

  $\Rightarrow$ implémentation obligatoire de la méthode `clone()`

Deux sous-interfaces notables de l'API PeerSim :

- EDProtocol : un protocole basé sur des événements
- Transport : un protocole d'envoi de messages entre deux nœuds avec simulation de latence

Classe EDSimulator

Caractéristiques

- Classe singleton (que du `static`)
- Contient la file d'événements
- Extrait les événements en tête de file et les délivre aux nœuds déstinataires

Méthode utile

```
static void add(long delay, Object event, Node node, int pid)
```

- Ajoute l'événement `event` dans la file
- qui sera délivré dans `delay` unités de temps
- au nœud `node`
- pour le protocole d'identifiant `pid`

Pour simuler l'envoi d'un message :

```
EDSimulator.add(45, new MonMessage(), node, pid)
```

Interface Control

Caractéristiques

- Représente un module de contrôle qui peut être invoqué :
  - à l'initialisation du système (construction d'une topologie du système, amorçage de l'application, etc.)
  - pendant la simulation périodiquement ou ponctuellement
  - à la fin de la simulation
- Permet de simuler :
  - Des événements périodiques du système
  - L'activité de la couche applicative
  - Des événements extérieurs : pannes, départs de nœuds, de liens, etc.
- Permet de monitorer le système

Méthodes à définir

`boolean execute()` appelée lors de l'invocation du contrôle :

- retourne `true` si la simulation doit être stoppée
- retourne `false` sinon

Le fichier de configuration

Caractéristiques

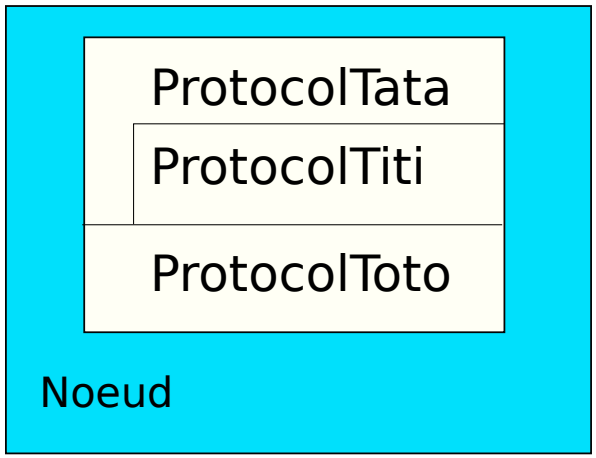
- Contient des couples clé/valeurs (properties Java)
- Spécifie les paramètres de simulation (taille du réseau, temps de simulation, graine du générateur aléatoire, etc.)
- Fixe les paramètres :
  - des différents protocoles (intervalle de maintenance, temps de latence, etc.)
  - des modules de contrôle (date de déclenchement)
- Fait le lien entre les différents protocoles (décrire le modèle en couches)

Fichier de configuration : les paramètres généraux

```
#date de fin de la simulation
simulation.endtime 50000

#taille du réseau en nombre de noeuds, REQUIS
network.size 10

#valeur de la graine du générateur aléatoire.
#par défaut: date courante
random.seed 45

#Classe implantant la PriorityQ. default : peersim.edsim.Heap
#simulation.eventqueue

#nombre de fois où l'expérience est exécuté, default : 1
#simulation.experiments

#fichier de redirection de la sortie standard.
#simulation.stdout
```

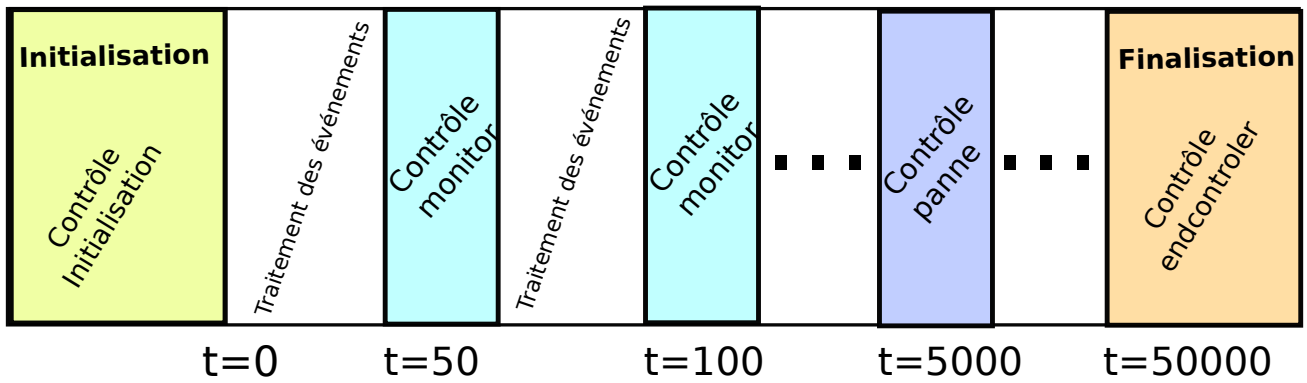
Fichier de configuration : les protocoles et leurs liens

```
protocol.protocole_toto ProtocolToto    # nom de la classe à charger
protocol.protocole_toto.attributX truc  # valeur de l'attribut x
protocol.protocole_toto.attributY 42    # valeur de l'attribut y

protocol.protocole_titi ProtocolTiti    # nom de la classe à charger
protocol.protocole_titi.attributToto protocole_toto

protocol.protocole_tata ProtocolTata    # nom de la classe à charger
protocol.protocole_tata.attributToto protocole_toto
protocol.protocole_tata.attributTiti protocole_titi
```



Fichier de configuration : les modules de contrôle

```
#definition des modules de controle d'initialisation
init.initialisation Initialisation      # nom de la classe à charger
init.initialisation.attributZ 78

#controles de simulation
control.monitor Monitor                 # nom de la classe à charger
control.monitor.from 0                  # date de début de repétition
control.monitor.until 10000             # date de fin de répétition
control.monitor.step 50                 # pas de répétition

control.panne PanneAleatoire            # nom de la classe à charger
control.panne.at 5000                   # module de controle ponctuel

#controle de finalisation
control.endcontroler EndControler       # nom de la classe à charger
control.endcontroler.at -1              # pas d'execution pendant la simu
control.endcontroler.FINAL              #  le controle doit s'executer après la fin
```



La classe Configuration

Caractéristiques

- Classe singleton (que du `static`)
- Permet d'accéder aux paramètres dans le fichier de configuration

Méthodes utiles

- `int getInt(String k)`, `long getLong(String k)`, `double getDouble(String k)`, `boolean getBoolean(String k)`, `String getString(String k)`
  renvoient la valeur associée à la clé `k`
- `int getPid(String k)` : si la valeur associée à la clé `k` est un nom de protocole, renvoie l'identifiant du protocole
- `int lookupPid(String name)` : renvoie le pid du protocole de nom `name`
- `String lookupPid(int pid)` : renvoie le nom du protocole associé à l'identifiant `pid`

Écrire une classe contrôle

Méthodologie

- Le constructeur doit avoir un seul argument de type String
- La valeur de l'argument est égale au préfixe de la clé du module dans le fichier de configuration
- Les attributs sont initialisés avec les méthodes de la classe `Configuration`

```
public class MonControl implements Control{
  private int a;
  public MonControl(String prefix){
    a=Configuration.getInt(prefix+".attributA");
  }
  public boolean execute(){ //code du module de controle ... }
}
```

Implique dans le fichier de configuration :

```
control.mon_control MonControl
control.mon_control.attributA 54
```

Écrire une classe Protocole

Méthodologie

- la même que pour les modules de contrôle
- Implémenter en plus la méthode `clone()`

```
public class MonProtocol implements Protocol{
  private final int my_pid, pid_subproto;
  public MonProtocol(String prefix){
    String tmp[]=prefix.split("\\.");
    my_pid=Configuration.lookupPid(tmp[tmp.length-1]);
    pid_subproto = Configuration.getPid(prefix+".subproto");

  }
  public Object clone(){ //code du clonage ... }
}
```

Implique dans le fichier de configuration :

```
protocol.mon_protocol MonProtocol
protocol.mon_protocol.subproto autre_protocol

protocol.autre_protocol MonAutreProtocol
```

Clonage des nœuds

Au lancement de la simulation :

- Création d'un nœud prototype contenant une instance de chaque protocole déclaré dans le fichier de configuration.
- Le prototype est cloné pour obtenir l'ensemble des nœuds.

Attention

- Pour chaque protocole différencier les attributs :
  - qui peuvent être partagés (ex : une variable globale en lecture seule)
  - qui doivent être copiés (ceux propre à chaque nœud)
- Tout copier = beaucoup de mémoire, Tout partager = simu erronée

```
private int[] tab;//partagé
private int[] tab2; // non partagé
public Object clone(){
      Object res=super.clone();
      res.tab2= new int[tab2.length];
      return res;
}
```

Interface EDProtocol

Caractéristiques

- Étend l'interface `Protocol`
- Représente un protocole basé sur des événements

Méthode à définir

`void processEvent( Node node, int pid, Object event );` appelée :

- lorsqu'un événement `event` (ex : un message)
- est délivré sur le nœud `node`
- pour le protocole d'identifiant `pid`
  (dans 99% des cas égal à l'id du protocole implémenté par la classe)

Le simulateur appelle cette méthode au dépilement de la file d'attente d'un événement concernant `node` pour le protocole `pid`

Dans le cadre de ce cours, c'est cette interface qui sera utilisée pour implémenter les algorithmes

Problème de l'envoi de messages

Inconvénients de la méthode `EDSimulator.add`

- Ajout direct depuis l'application dans la file d'événement
  $\Rightarrow$ dépendance du code applicatif avec le simulateur
- Obligation de définir au moment de l'envoi la date de réception
  $\Rightarrow$ API non réaliste
  $\Rightarrow$ latence réseau simulée dans le code applicatif

Il serait intéressant que l'envoi de message se fasse via une méthode `Send` sans soucier de la simulation réseau

$\Rightarrow$ Utilisation d'un protocole de transport

Interface Transport

Caractéristiques

- Étend l'interface `Protocol`
- Permet la simulation d'envoi de messages avec une latence réseau aléatoire ou prédéfinie

Méthode à définir

- `void send(Node src, Node dest, Object msg, int pid);` : simule l'envoi d'un message entre les nœuds `src` et `dest` pour le protocole `pid`
- `long getLatency(Node src, Node dest)` : retourne la latence réseau entre les nœuds `src` et `dest`

Protocoles de transport déjà fournis par PeerSim

UniformRandomTransport

- Délivre les messages avec une latence comprise entre une borne min et une borne max tirée selon une loi aléatoire uniforme
- Deux paramètres à renseigner dans le fichier de conf :

```
protocol.tr UniformRandomTransport
protocol.tr.mindelay 10
protocol.tr.maxdelay 70
```

UnreliableTransport

- Couche transport simulant des canaux non fiables
- Décore une couche transport existante
- Un message n'est pas envoyé avec une probabilité paramétrable

```
protocol.tr_unreliable UnreliableTransport
protocol.tr_unreliable.transport tr
protocol.tr_unreliable.drop 0.5
```

Construire une topologie avec l'interface `Linkable`

Caractéristiques

- Permet de stocker un vecteur de voisins par nœud
- Convient uniquement aux topologies statiques
  $\Rightarrow$ impossible de supprimer un lien en cours de simulation
- Protocole fourni implémentant cette interface : `IdleProtocol`
- Peut être utiliser pour manipuler des graphes
- Découple l'application de la topologie

Méthodes de l'interface `Linkable`

- `int degree()` : nombre de voisins
- `Node getNeighbor(int i)` : retourne le voisin d'indice i
- `boolean addNeighbor(Node ngb)` : ajouter un nouveau voisin
- `boolean contains(Node ngb)` : tester le voisinage
- `void onKill()` : appelée lorsque le nœud passe à l'état `DEAD`.

Exemple

Scénario :

- On initialise 5 nœuds organisés dans une topologie quelconque
- À l'initialisation, le nœud d'identifiant 0 diffuse un message "hello" à un de ses voisins.
- À la réception, un nœud affiche le message avec l'id de son expéditeur
- Si le récepteur n'a jamais envoyé de message, il diffuse à ses voisins un message "hello"

Composition du système

- Un protocole de transport (UniformRandomTransport)
- Un protocole applicatif de helloWorld
- Un protocole de topologie Applicative (IdleProtocol)
- Un module d'initialisation (déclenche l'envoi par le nœud 0 du message "hello" et construit la topologie).

Message du protocole applicatif

```
public  class HelloMessage {

 public final String content;
 public final long idsrc;
 public final long iddest;

 public HelloMessage(String content, long s, long d){
   this.content=content;
   this.iddest=d;
   this.idsrc=s;
 }
}
```

Protocole Applicatif : attributs et constructeur

```
public class HWprotocol implements EDProtocol{

 private final int my_pid;
 private final int topology_pid;
 private final int transport_pid;

 private boolean already_sent=false;

 public  HWprotocol(String prefix) {
  String tmp[]=prefix.split("\\.");
  my_pid=Configuration.lookupPid(tmp[tmp.length-1]);
  topology_pid=Configuration.getPid(prefix+".topo");
  transport_pid = Configuration.getPid(prefix+".tr");
 }

 public Object clone(){
  HWprotocol ap = null;
  try { ap = (HWprotocol) super.clone(); }
  catch( CloneNotSupportedException e ) {} // never happens
  return ap;
 }
```

Protocole Applicatif : API

```
public void sendToNeighbor(Node host, Node ngb, String mess){
 Transport tr = (Transport) host.getProtocol(transport_pid);
 HelloMessage msg = new HelloMessage(mess, host.getID(), ngb.getID())
 tr.send(host, ngb,msg, my_pid);
 already_sent=true;
}
 public void receiveMessage(Node host, HelloMessage msg){
  System.out.println(host.getID()+" recv from "+msg.idsrc+" : "
        +msg.content);
  if(!already_sent){
   Linkable topo = (Linkable) host.getProtocol(topology_pid);
   for(int i = 0; i< topo.degree();i++ )
     sendToNeighbor(host, topo.getNeighbor(i),"Hello");
  }
 }
 public void processEvent(Node host, int pid, Object event) {
  if(event instanceof HelloMessage && pid == my_pid){
   HelloMessage msg=(HelloMessage)event;
   receiveMessage(host, msg);
  }else
   throw new RuntimeException("Wrong Event");
 }
```

Module d'initialisation

```
public class HWinit implements Control{
 private final int topology_pid;
 private final int app_pid;
 public  HWinit(String prefix) {
  topology_pid=Configuration.getPid(prefix+".topo");
  app_pid = Configuration.getPid(prefix+".app_pid");
 }
 public boolean execute() {
  for(int i=0;i<Network.size();i++){
   Node n =  Network.get(i);
   IdleProtocol topo = (IdleProtocol)n.getProtocol(topology_pid);

   Node ngb_d = Network.get((i+1) % Network.size());
   topo.addNeighbor(ngb_d);
   Node ngb_g = Network.get(i==0?Network.size()-1:i-1);
   topo.addNeighbor(ngb_g);

   if(n.getID() == 0){
    HWprotocol app_zero = (HWprotocol) n.getProtocol(app_pid);
    app_zero.sendToNeighbor(n, ngb_d, "Hello");
   }
  }
  return false;
 }
```

Fichier de configuration

```
simulation.experiments 1

random.seed 10

simulation.endtime 1000

network.size 5

protocol.transport UniformRandomTransport
protocol.transport.maxdelay 10
protocol.transport.mindelay 1

protocol.topology IdleProtocol

protocol.helloring HWprotocol
protocol.helloring.topo topology
protocol.helloring.tr transport

init.helloinit HWinit
init.helloinit.topo topology
init.helloinit.app_pid helloring
```

Pour finir

The Fantastic Manual

- Le code de Peersim est bien documenté.
- La Javadoc est disponible ici : http://peersim.sourceforge.net/doc/index.html
- Le fonctionnement général de Peersim, notamment l'utilisation du fichier de configuration est détaillé ici : http://peersim.sourceforge.net/tutorial1/tutorial1.pdf
- Le fonctionnement du modèle à événements discrets de Peersim est illustré avec des exemples là : http://peersim.sourceforge.net/tutorialed/tutorialed.pdf