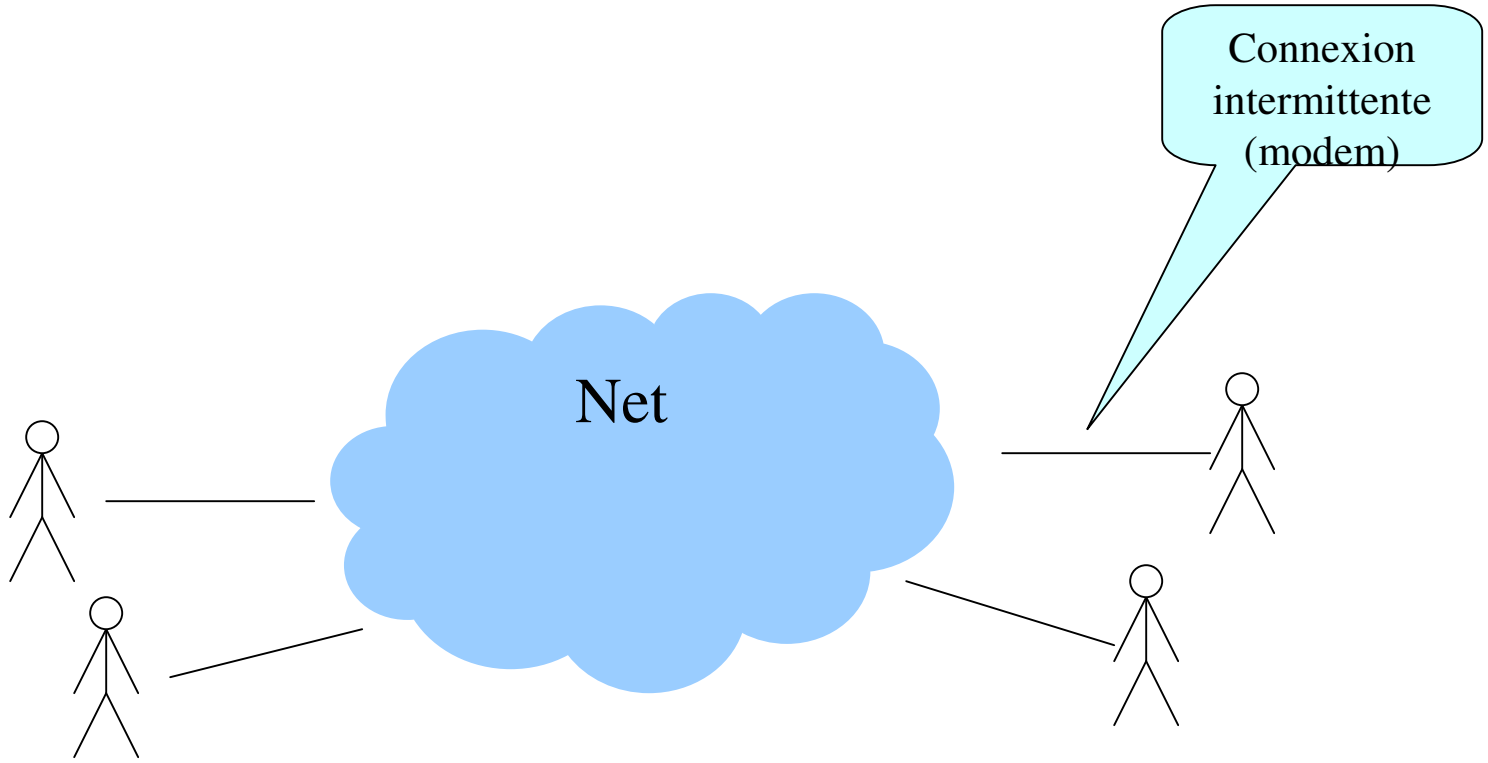
Didier DONSEZ
Université Joseph Fourier
IMA –IMAG/LSR/ADELE
Didier.Donsez@imag.fr
Didier.Donsez@ieee.org

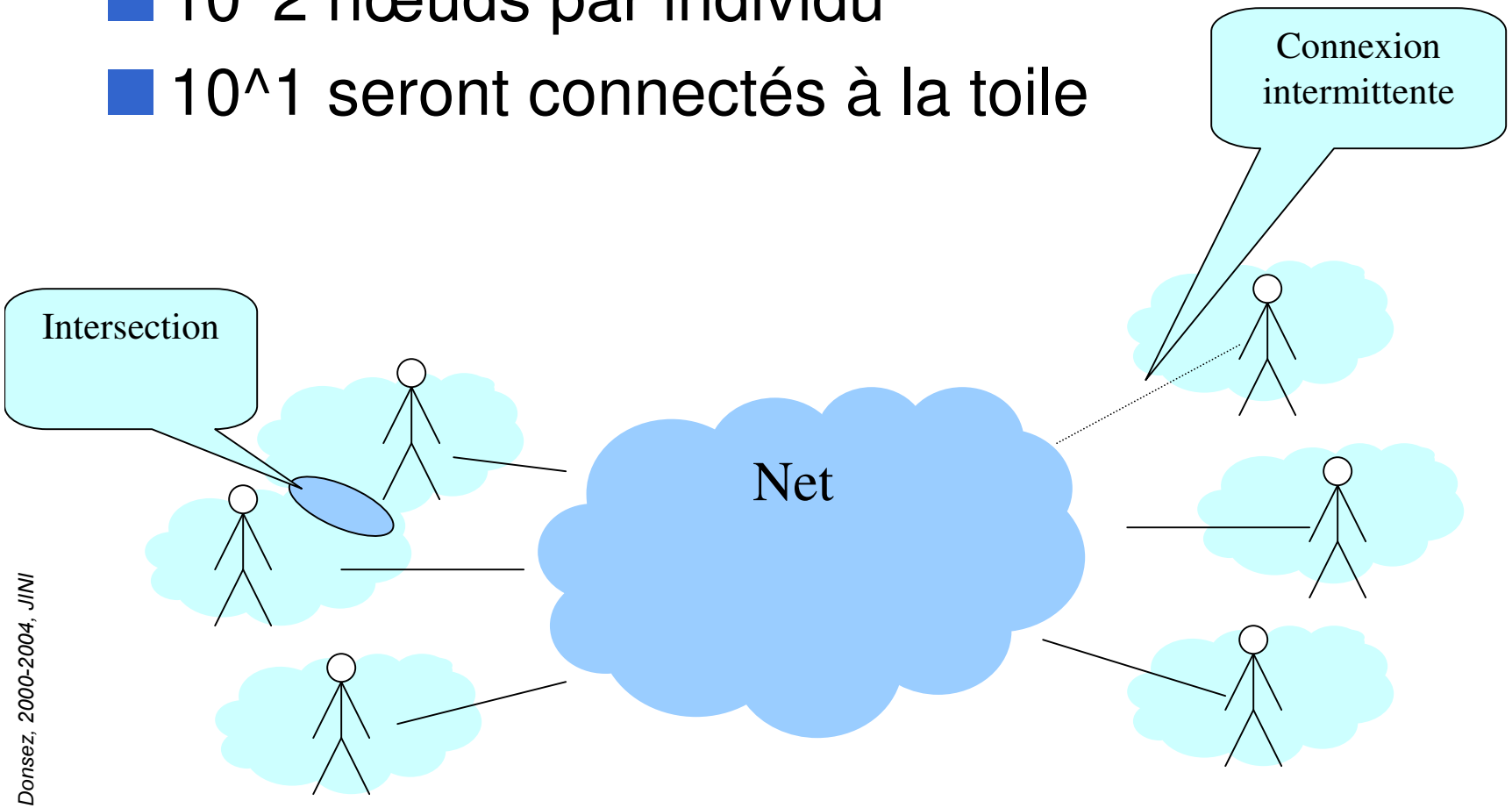
La toile : maintenant

- 10^10 nœuds
- 10^1 nœuds par individu
- 10^0 sont reliés directement à la toile



La toile : demain (matin)

- 10^12 nœuds
- 10^2 nœuds par individu
- 10^1 seront connectés à la toile



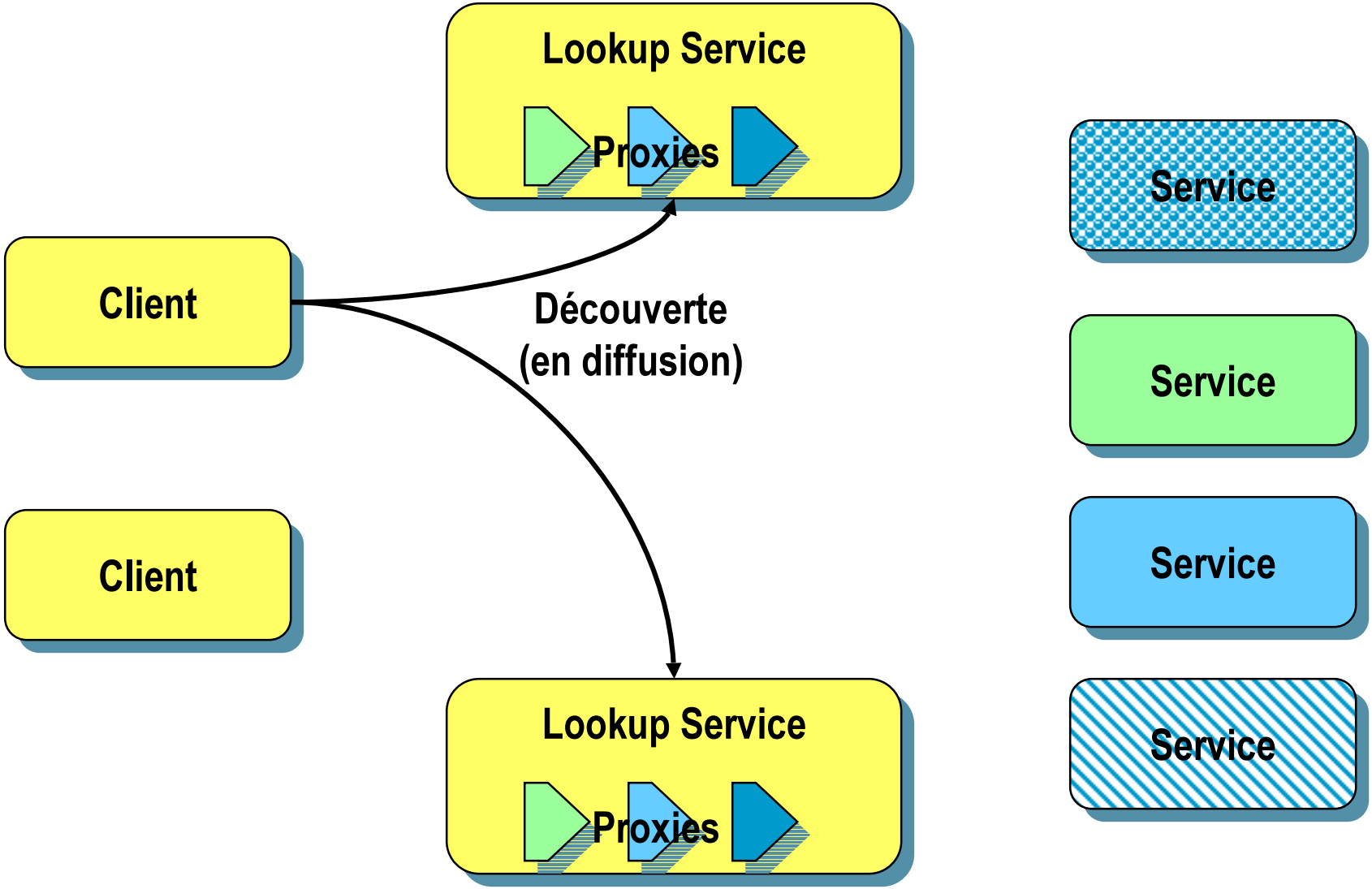
Motivations de JINI : Plug-and-Participate

- Support pour le développement d'applications distribuées
  - Spontanées
    - les fournisseurs de service apparaissent et disparaissent de l'environnement réseau
  - Tolérantes aux pannes
    - L'environnement réseau peut subir des pannes transitoires pouvant former des partitions de réseaux
  - Peer To Peer (P2P)
    - Tout nœud est client, serveur, et indexeur
- Cible
  - Simplification des interfaces et de la connexion entre ordinateurs nomades, enfouis, …
    - PC, PDA, téléphone mobile, TV, STB, Console de Jeux, Digital Camera, HiFi, Imprimantes, Fax, Alarmes, GPS, Domotique, …

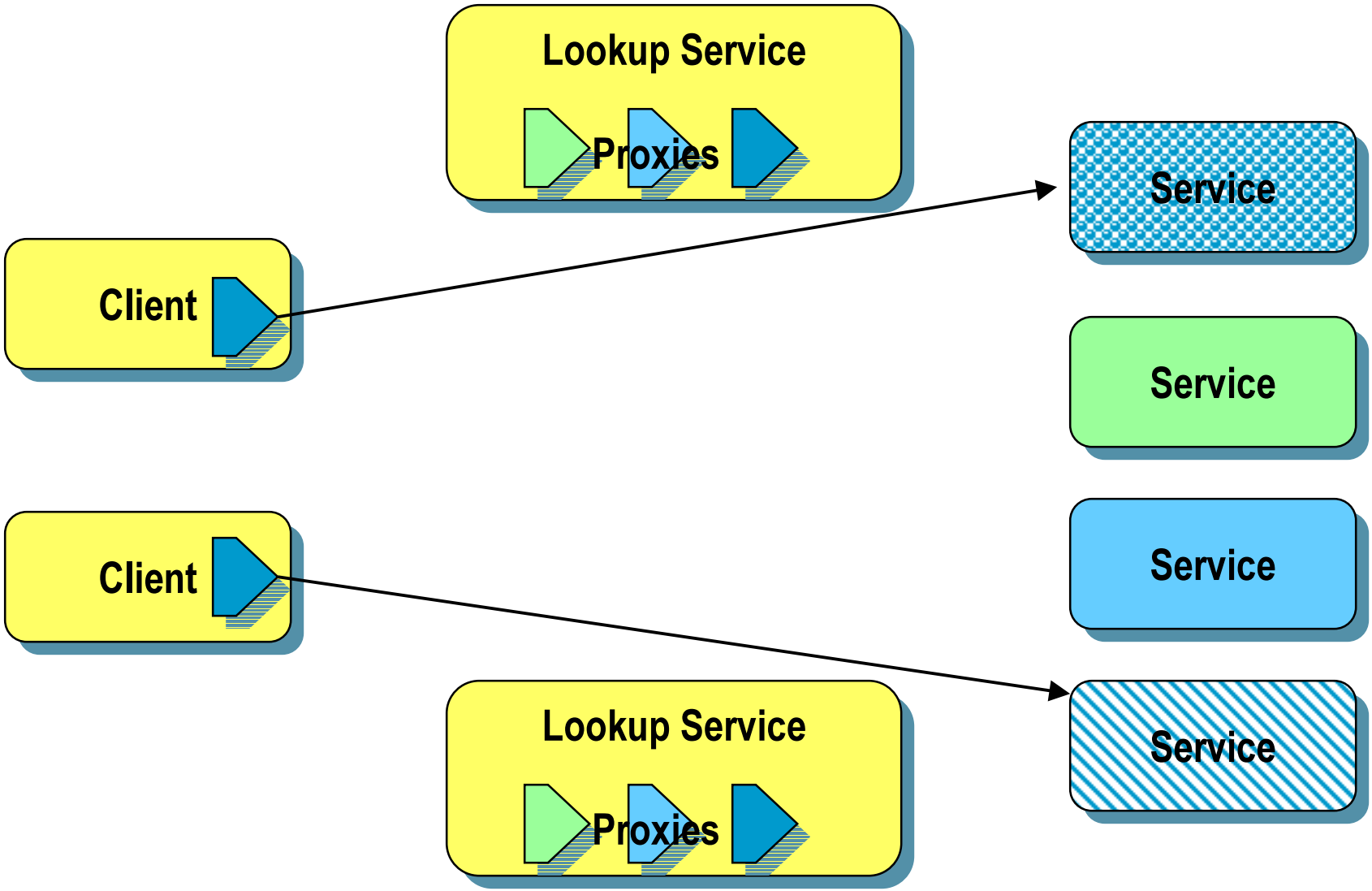
Architecture

- Service
  - Prepare un proxy d'accès au service
  - Recherche un service de lookup
  - Enregistre le proxy auprès du service de lookup
- Proxy
  - Objet sérialisable qui assure la connexion au Service et les invocations de méthodes
    - Peut être un Stub RMI ou tout autre chose (contient un état, utilise un protocole particulier)
- Lookup
  - Les services enregistrent leur Proxys auprès des Lookups
- Client
  - Interogge (en diffusion) tous les lookups voisins pour obtenir des listes de proxys
  - Invoque les méthodes sur un proxy (en utilise un autre en cas d'exception)

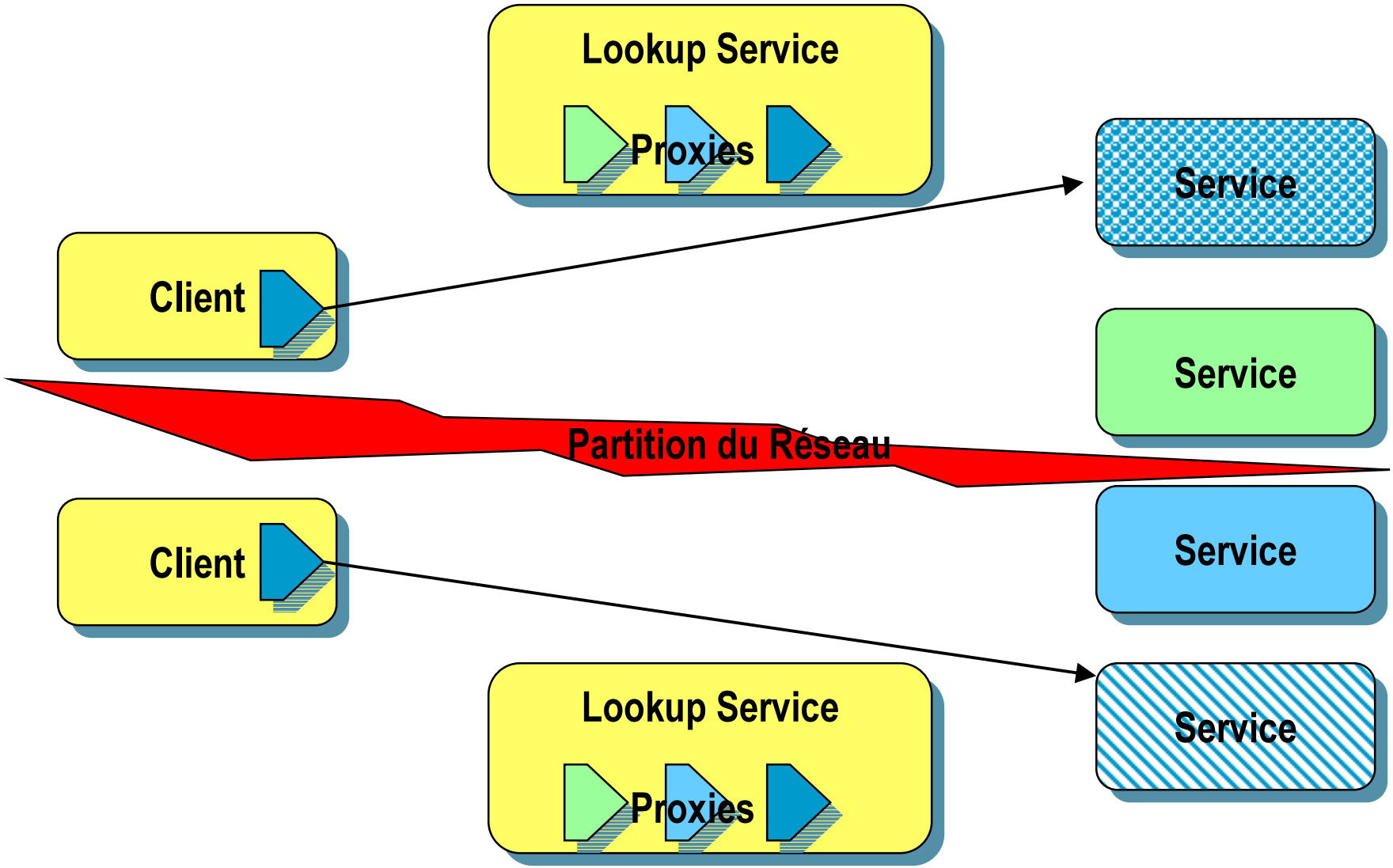
Architecture



Architecture



Architecture



Principe discovery-join-lookup du JINI (i)

- Rappel RMI
  - Naming RMI
    - Un seul serveur de nom associe un nom (chaîne de caractère) au stub d'un service (objet) distant (actif ou activable).
  - Lookup RMI
    - Le client s'adresse à ce serveur de nom pour récupérer ce stub et invoquer des méthodes sur l'objet distant actif
- Inconvénients
  - Le client doit connaître le serveur de nom
    - Adresse IP/Nom DNS et port (1099 par défaut)
  - Le serveur peut tomber en panne
    - En cas de reprise, le service ne sait plus qu'il n'est pas enregistré
  - Le nom correspond à un seul service (objet) distant
    - Alors que plusieurs objets pourraient rendre le même service
  - Le nommage est limité à la chaîne de caractère
  - La recherche d'un nom est accessible à tout client

Principe discovery-join-lookup de JINI (ii)

Tolérance aux pannes du nommage

- Partition de réseau
  - Certains entités s'en rendent compte, d'autres non
- Arrêt brutal d'un serveur de nom / Reprise
- Arrêt brutal d'un service / Reprise

Solutions

- Emission régulière de message « I'm alive »
  - Mode multicast
- Redondance du service de nommage
  - La cohérence forte n'est pas nécessaire

Principe discovery-join-lookup de JINI (iii)

- Nommage
  - Fortement typé
  - Attributs
  - Groupes
- Service
  - Plusieurs providers (rmid ou server) peuvent fournir le même service (implémetations différentes)
- Solutions
  - Fortement typé
    - La recherche se fait par rapport à une classe/interface
  - Entrées
  - Groupes
    - Un service/client peut participer à plusieurs groupes
  - Stub
    - RMI ou Autre

Rôle du Proxy

- Objet sérialisé qui peut être chargé pour accéder au service

Structures du Client et du Serveur

- Serveur
  - prepare for discovery Discovering a lookup service
  - discover a lookup service Discovering a lookup service
  - create information about a service Entry objects
  - export a service Service registration
  - renew leasing periodically Leasing
- Client
  - prepare for discovery Discovering a lookup service
  - discover a lookup service Discovering a lookup service
  - prepare a template for lookup search Entry objects and Client search
  - lookup a service Client searchcall the service

La Découverte Unicast Synchrone

- Classe net.jini.core.discovery.LookupLocator
  - Hypothèse : On connaît le l'hôte et le port d'un LS
  - Constructeur
    - LookupLocator(java.lang.String url) throws MalformedURLException;

      jini://host:port

      jini://host (port=4160)
    - LookupLocator(java.lang.String host,int port);
  - Méthode
    - ServiceRegistrar getRegistrar()

La Découverte Multicast Asynchrone (i)

- Hypothèse :
  - On ne connaît les <hôte,port> des LS voisins
  - La découverte de LS peut être restreinte à des groupes
- Classes et Interfaces
  - net.jini.core.discovery.LookupDiscovery
    - L'application enregistre un DiscoveryListener
  - net.jini.core.discovery.DiscoveryListener
    - Les méthodes discovered() et discarded() sont invoquées de manière asynchrone lors de l'apparition et la disparition de LS
  - net.jini.core.discovery.DiscoveryEvent
    - Contient les ServiceRegistrar des LS qui apparaissent et disparaissent

La Découverte Multicast Asynchrone (ii)

- Classe net.jini.core.discovery.LookupDiscovery
  - Permet à l'application d'enregistrer un DiscoveryListener
  - Constructeur
    - LookupDiscovery(String [] groups);

      La découverte est restreinte aux LS qui servent un des groupes listés

      ALL_GROUPS et null signifient qu'il y a pas de restriction

      NO_GROUPS et { } rend inactive la découverte de LS
  - Méthode
    - public void addDiscoveryListener(DiscoveryListener l)

      Ajoute un objet qui implémente DiscoveryListener
    - setGroups(String [] groups)

      Permet d'ajouter des groupes pour la découverte

La Découverte Multicast Asynchrone (iii)

- Interface net.jini.discovery.DiscoveryListener
  - Permet de notifier de manière asynchrone l'application de l'arrivée ou la disparition de LS.
  - Méthodes
    - public void discovered(DiscoveryEvent e);

      Invoqué quand un ou plusieurs LS sont découverts
    - public void discarded(DiscoveryEvent e);

      Invoqué quand un ou plusieurs LS ont disparu
- Classe net.jini.discovery.DiscoveryEvent
  - public ServiceRegistrar[] getRegistrars();
    - Tableau des ServiceRegistrar des LS découverts ou disparus

L'interrogation d'un LS

- Classe abstraite net.jini.core.lookup.ServiceRegistrar
  - Rôle : Proxy vers le LS
- Méthodes
  - Enregistrement de service (par son Serveur)
    - public ServiceRegistration register(ServiceItem item, long leaseDuration) throws java.rmi.RemoteException
  - Recherche d'un service (par un Client)
    - public java.lang.Object lookup(ServiceTemplate tmpl) throws java.rmi.RemoteException

      Retourne un seul service
    - public ServiceMatches lookup(ServiceTemplate tmpl, int maxMatches) throws java.rmi.RemoteException;

      Retourne jusqu'à maxMatches services

L'enregistrement d'un Service par le Serveur

- Méthodes register() de ServiceRegistrar
- Classes
  - net.jini.core.lookup.ServiceItem
    - Caractérise le service à enregistrer

      ServiceID serviceID,
      - » Identifiant d'un service

      java.lang.Object service,
      - » Proxy vers le service

      Entry[] attributeSets
      - » Tableau d'entrées caractérisant le service
  - net.jini.core.lookup.ServiceRegistration
    - Permet de modifier les caratéristiques (Entry[]) du service après l'enregistrement
    - Permet de prolonger le bail (Lease)
    - Permet de récupérer le ServiceID

La Recherche d'un Service par le Client

- Méthodes lookup() de ServiceRegistrar
- Classes
  - net.jini.core.lookup.ServiceTemplate
    - Définit le profil des services recherchés

```
ServiceTemplate(
    ServiceID serviceID,
        » Identifiant(universel) d'un service (128 bits générés aléatoirement)
    java.lang.Class[] serviceTypes,
        » Ensemble des classes (et sous-classes) recherchées
    Entry[] attrSetTemplates
);
```

  - net.jini.core.lookup.ServiceMatches

```
public class ServiceMatches {
public ServiceItem[] items;
public int totalMatches ;
}
```

Restriction de la Recherche

- ServiceID
  - ==null : tout service
  - !=null : le service a qui le LS a donné ce numéro lors de l'enregistrement
- Class[]
  - ==null : tout service
  - La classe du service recherché doit être une des classes du tableau ou une de leur sous classes
- Entry[]
  - ==null : tout service
  - Le service recherché doit avoir une des Entry qui vérifie avec equals() les Entry du tableau

Exemple

```
public interface PrinterIntf implements java.rmi.Remote {
    public void print(Document doc) throws java.rmi.RemoteException;
}
public class abstract Printer implements PrinterIntf {
    public void print(Document doc) throws java.rmi.RemoteException;
}

public class PaperQuality extends Entry {
    public String quality;
    public PaperQuality(String quality) {this.quality=quality};
}
public class PaperSize extends Entry {
    public String size;
    public PaperSize(String size) {this.size=size};
}
```

Exemple d'enregistrement

```
LookupLocator lookup = new LookupLocator("jini://hostreg");
ServiceRegistrar registrar = lookup.getRegistrar();

Printer colordeskPrinterService = new ColorDeskPrinterService(); // objet UnicastObject
Entry[] colordeskPrinterAttr={ new PaperQuality("photo"), new PaperSize("A4") };
ServiceItem itemcolor=new ServiceItem(null, colordeskPrinterService , colordeskPrinterAttr);
ServiceRegistration regcolor = registrar.register(itemcolor, Lease.ANY); // bail fixé par le LS

Printer bwdeptPrinterService = new BWDeptPrinterService(); // objet UnicastObject
Entry[] bwdeptPrinterAttr={     new PaperQuality("normal"),
                                new PaperSize("A4"), new PaperSize("US letter") };
ServiceItem itembw=new ServiceItem(null, colordeskPrinterService , colordeskPrinterAttr);
ServiceRegistration regbw = registrar.register(itembw, 10*60*1000); // 10 minutes

System.out.println("Lease of service "+ regbw.getServiceID() +" expires at: «
            + regbw.getLease().getExpiration() - System.currentTimeMillis() + " milliseconds
    from now");
```

Exemple de recherche

```
LookupLocator lookup = new LookupLocator("jini://hostreg");
ServiceRegistrar registrar = lookup.getRegistrar();
// Selectionne toutes les imprimantes mais n'en utilise qu'une
Class[] printerClasses=new Class[1]; printerClasses[0]=Printer.class;
ServiceTemplate allPrinterTemplate=new ServiceTemplate(null, printerClasses, null)
Printer printer = (Printer)registrar.lookup(allPrinterTemplate);
printer.print(new TextDocument("report.txt"));
// Selectionne les imprimantes dont les Entry contient PaperSize("A4") ou PaperSize("A3")
Entry[] A4A3PaperSizeAttr={ new PaperSize("A4"), new PaperSize("A3") };
ServiceTemplate A4A3PrinterTemplate=new ServiceTemplate(null, printerClasses, A4A3PaperSizeAttr)
ServiceMatches matches = (Printer)registrar.lookup(A4A3PrinterTemplate, 10);
for (int n = 0; n < matches.items.length; n++) {
      Printer a4a3printer = (Printer) matches.items[n].service;
      if (a4a3printer != null) {a4a3printer.print(new Document("report.txt"));}
// Selectionne les imprimantes couleur mais n'en utilise qu'une
Class[] colorPrinterClasses=new Class[1]; colorPrinterClasses[0]=ColorPrinter.class;
ServiceTemplate allPrinterTemplate=new ServiceTemplate(null, colorPrinterClasses, null)
Printer colorPrinter = (Printer)registrar.lookup(colorPrinterTemplate);
colorPrinter.print(new JPEGDocument("family.jpg"));
```

Exemple de Service JINI

- L'interface distante Jtime
  - Remarque : Le proxy sera un stub RMI

```
public interface JTime extends java.rmi.Remote {
 public long getTime() throws java.rmi.RemoteException;
}
```

- L'objet distant JTimeImpl
  - Préférable en objet activable

```
public class JTimeImpl
    extends UnicastRemoteObject
     implements JTime, ServiceIDListener, Serializable {
 public JTimeImpl() throws RemoteException { super (); }
 public long getTime() throws RemoteException {
     return(System.currentTimeMillis()); }
 public void serviceIDNotify (ServiceID serviceId) { }
}
```

Exemple de Service JINI

Le serveur

- Démarre l'objet et le « publie »

```
public class JTimeServer {
  public static void main (String[] args) {
  try {
   System.setSecurityManager (new RMISecurityManager ());
   Entry[] aeAttributes = new Entry[1];
   aeAttributes[0] = new Name("Time");
   aeAttributes[1] = new Name("Temps");
   aeAttributes[2] = new Name("Tiempo");
   JTime jtime = new JTimeImpl();
   Joinmanager joinmanager = new JoinManager(jtime,aeAttributes,
                                                    jtime,new LeaseRenewalManager());
  } catch(Exception e) { System.err.println(e); }
}
```

Exemple d'un client unicast

```
class JTimeClient {
 public static void main (String[] args) {
try {
  System.setSecurityManager (new RMISecurityManager ());
  LookupLocator lookup = new LookupLocator ("jini://localhost");
  String sHost = lookup.getHost ();    int iPort = lookup.getPort ();
  ServiceRegistrar registrar = lookup.getRegistrar ();
  ServiceID id = registrar.getServiceID ();
  Entry[] aeAttributes = new Entry[1];
  aeAttributes[0] = new Name ("Time");
  ServiceTemplate template = new ServiceTemplate (null, null, aeAttributes);
  TimeServer timeServer= (TimeServer) registrar.lookup (template);
  if (timeServer  instanceof TimeServer) {
   long startTime = timeServer.getTime();  System.out.println ("TimeClient: Time at server is " + startTime);
   long stopTime = System.currentTimeMillis();  System.out.println ("TimeClient:: Time at client is " + stopTime);
   float difference= stopTime-startTime;
   System.out.println ("TimeClient:: Time for message to traverse the net is --->" + difference + " msecs<---");
  }
 } catch (Exception e) { System.out.println ("TimeClient: MyClient.main() exception: " + e); }  } }
```

Exemple
Scrutation des services JINI

```
class JiniLookupSnooper {
public static void main (String[] args) {
 try {
  System.setSecurityManager (new RMISecurityManager ());
  LookupLocator lookup = new LookupLocator ("jini://localhost");
  System.out.println ("LookupLocator = " + lookup);
  System.out.println ("LookupLocator.host = " + lookup.getHost());
  System.out.println ("LookupLocator.port = " + lookup.getPort());
  ServiceRegistrar registrar = lookup.getRegistrar();
  System.out.println ("ServiceRegistrar = " + registrar);
  System.out.println ("ServiceID = " + registrar.getServiceID ());
  ServiceMatches matches = registrar.lookup(new ServiceTemplate (null, null, null),50);
  System.out.println ("ServiceMatches = " + matches);
  System.out.println ("num matches = "+ matches.totalMatches);
  for (int i = 0; i < matches.totalMatches; i++) {
   System.out.println ("svc item " + i + ": " + matches.items[i]);
   System.out.println ("svc object " + i + ": "+ matches.items[i].service);
  }
  System.out.println ("*---------------------------------------*");
 } catch (Exception e) { System.out.println("main(): Exception " + e); }  }  }
```

Type de Proxy de Service

- Type 1 : Stub RMI
  - Généré par rmic
  - Utilise donc TPC/IP des RMI
- Type 2 : Stub RMI + Etat
  - objet sérialisable qui contient des informations d'état et un stub RMI
- Type 3 : protocole propre
  - Protocole réseau autre que TCP
    - HTTP, UDP, propriétaire (HP JetSend,…), …
  - Encodage/Décodage des méthodes
  - Utilise par les péripheriques propriétaires

Les protocoles de Lookup

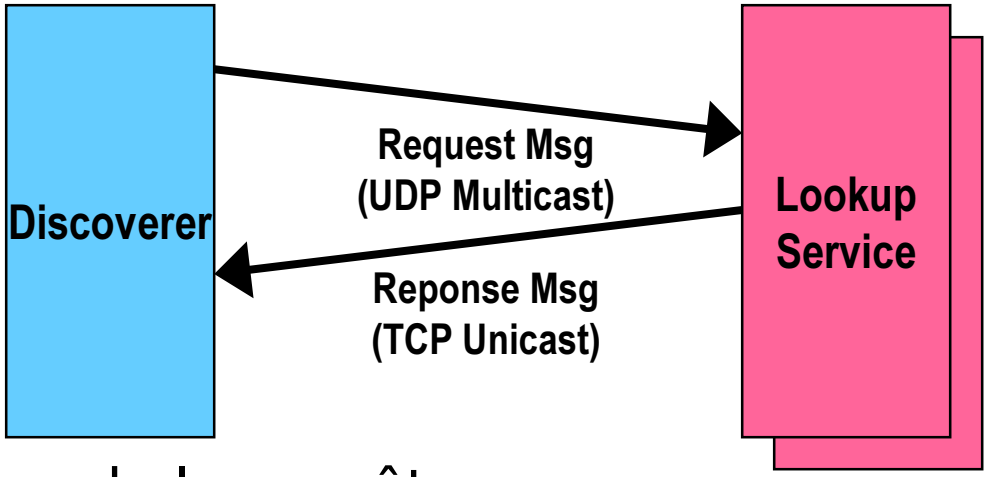
3 protocoles entre

Discovery Entity (DE) et Lookup Server (LS)

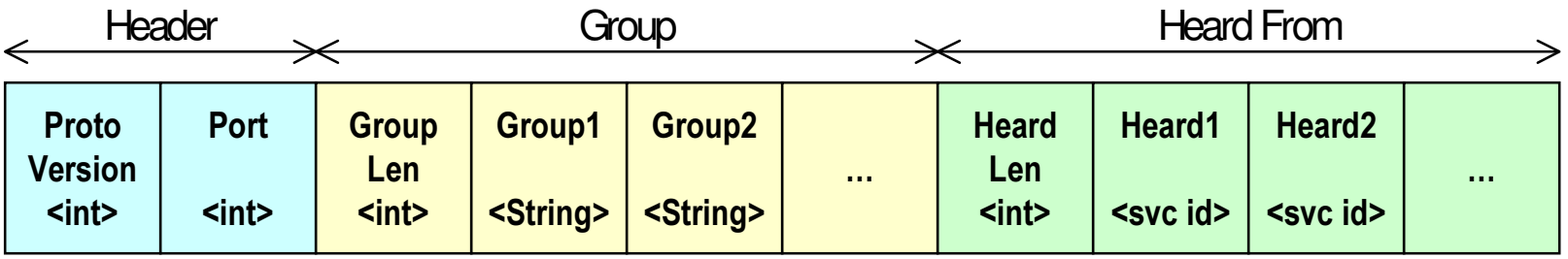
- Multicast Request Protocol
- Multicast Announcement Protocol
- Unicast Discovery Protocol (TCP)

Multicast Request Protocol

- But: Découverte par DE des LS voisins
  - Au démarrage du DE

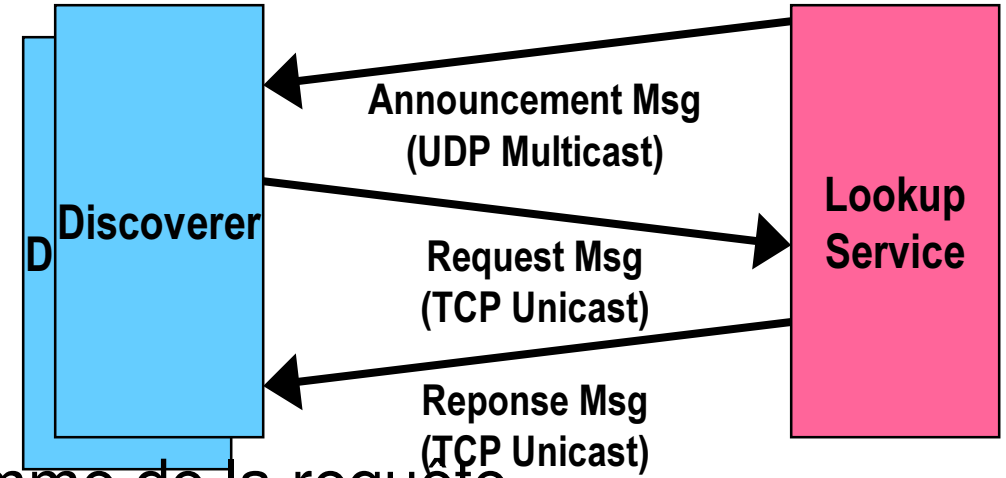


- Datagramme de la requête

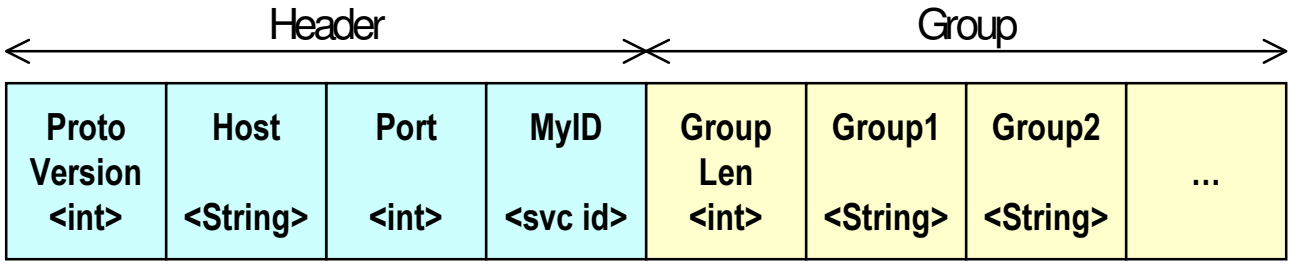


Multicast Announcement Protocol

- But : Un LS annonce sa présence
  - Startup du LS puis régulier (120sec) au cas où le réseau tombe



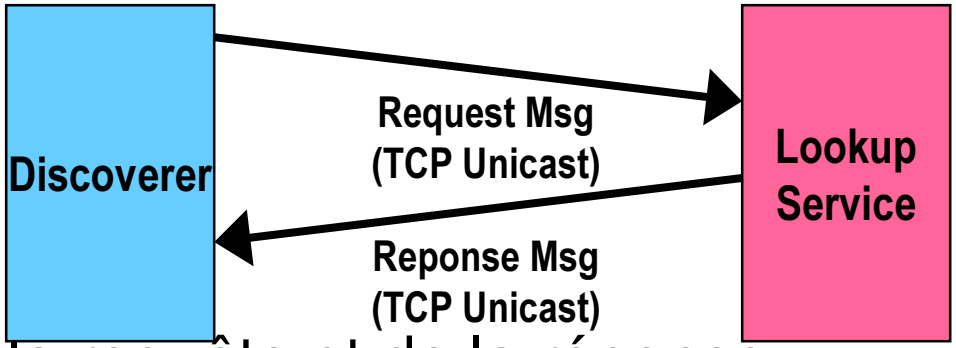
- Datagramme de la requête



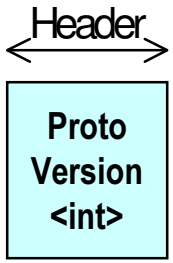
Unicast Discovery Protocol (TCP)

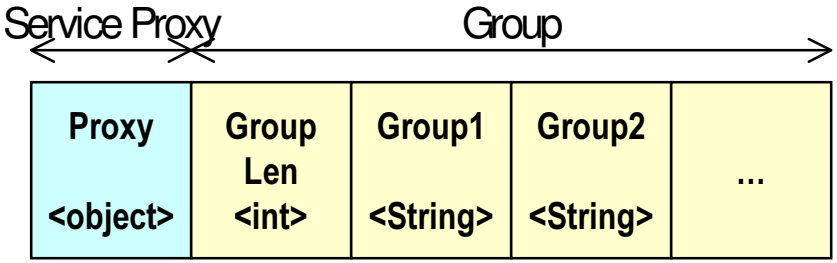
Connexion initiée par le LS en réponse au Multicast Request

- Connexion initié par le DE (quand DE connaît <hote,port> d'un LS
- DE envoie la version du protocole à LS
- LS répond à DE avec un liste de groupes



- Format de la requête et de la réponse





Constantes des protocoles

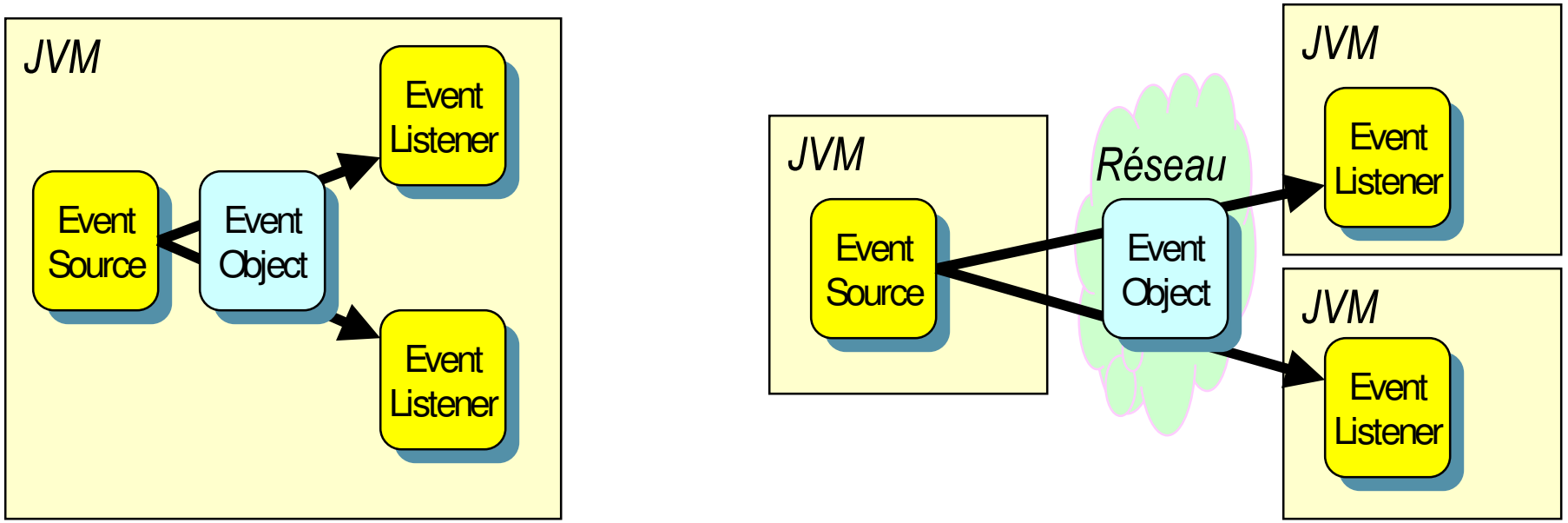
- Multicast Protocols
  - LAN IP Broadcast (comme DHCP ou BOOTP)
  - IP Multicast (recommendé)
    - 224.0.1.85:4160 pour Multicast Request Protocol
    - 224.0.1.84:4160 pour Multicast Announcement Protocol
    - TTL (Time-to-Live) 15 par défaut

      Propriété net.jini.discovery.ttl
    - Datagramme d'au maximum 512 octets
- Unicast Discovery Protocol
  - TCP : le port est aussi 4160 (=*0xCAFE – 0xBABE*)
- Intervalle de réémission
  - Multicast Announcement Protocol : 120 sec par défaut

Services JINI

- Lookup
  - classes utilitaires
    pour la découverte/interrogation de serveurs de Lookup
- Distributed Events
  - Intégration aux JavaBeans
- Transaction
  - Validation à deux phases
  - Transactions imbriquées (Nested)
- JavaSpace
  - Espace de données persistant distribué

Distributed Events



- Intégration au modèle des JavaBeans

Concurrence

- Salutation
  - Voir article Spectrum

Bibliographie JINI

Livres JINI

- Scott Oaks & Henry Wong , "Jini in a Nutshell", 1st edition March 2000, ISBN 1-56592-759-1, 416 pages, Ed Oreilly
- W. Keith Edwards , « Core Jini », June 1999, Prentice Hall; ISBN: 013014469X
- Ken Arnold, Bryan Osullivan, Robert W. Scheifler, Jim Waldo, Ann Wollrath, Bryan O'Sullivan , « The Jini(TM) Specification (The Jini(TM) Technology Series) », 1 edition (June 1999), Addison-Wesley Pub Co; ISBN: 0201616343
- Professional JINI, Ed Wrox, 2000

Webographie JINI

- Sites
  - http://www.sun.com/jini
  - http://www.kedwards.com/jini
  - http://www.jini.org
  - http://www.artima.com/jini
- Tutorial
  - Jan Newmarch's Guide to JINI Technologies
    - http://pandonia.canberra.edu.au/java/jini/tutorial/Jini.xml

      Le plus à jour et complet
  - Gopalan Suresh Raj
    - http://www.execpc.com/~gopalan/java/jini.html
  - Noel Enete's ``Nuggets'' tutorial
    - www.enete.com/download/#_nuggets_
  - Eran Davidov's timeservice example
    - www.artima.com/jini/resources/timeservice.html