IPv6 et EGEE : Migration d'un système d'information complexe vers IPv6, gLite

Etienne Dublé, Xavier Jeannin

IPv6 et EGEE : Migration d'un système d'information complexe vers IPv6, gLite

Etienne DUBLE
UREC (Unité Réseaux du CNRS)
Univ. P. & M. Curie Tour 66/65 - 4ème étage – 4 place de Jussieu 75252 Paris Cedex 05

Xavier Jeannin
UREC (Unité Réseaux du CNRS)
Univ. P. & M. Curie Tour 66/65 - 4ème étage – 4 place de Jussieu 75252 Paris Cedex 05

Résumé

La grille Enabling Grids for E-sciencE (EGEE, [1]) est gérée par un middleware nommé gLite qui constitue un véritable système d'information (authentification et autorisation, stockage, gestion du déplacement des données et des jobs, ...). A titre d'indication, le middleware est déployé sur 300 sites et traite 12 000 000 de jobs par mois. La croissance d'EGEE, le besoin d'adresses publiques, le besoin d'interconnexion avec les autres grilles et les recommandations de la commission européenne ont poussé le projet à travailler sur le passage à IPv6.

Ce document présente un retour d'expérience de l'équipe de travail sur IPv6 de l'activité SA2 d'EGEE. Porter sur IPv6 un ensemble de logiciel de cette taille est une tâche délicate. Bien entendu on ne peut espérer que tous les modules migreront en même temps. Compte tenu du déploiement d'IPv6 dans les NRENs, la solution dual-stack nous est apparue comme la meilleure. Pour démontrer la faisabilité nous avons porté un premier module et implémenté les deux premiers sites dual-stack de la grille. Nous avons aussi fourni aux développeurs une série d'études dont la synthèse se trouve dans cet article. Comment tester la compatibilité IPv6 d'un programme ? Comment porter un serveur en C/C++, Java, Python, PERL ? Toutes ces questions ont trouvé réponse et des résultats originaux ont été mis en évidence.

Enfin l'UREC a développé un outil original de vérification de compatibilité IPv6, dénommé IPv6 CARE : IPv6 Compliance Automatic Runtime Experiment. IPv6 CARE analyse l'exécution d'un programme et fournit un rapport sur les appels de fonctions relatifs au réseau. Un grand avantage d'IPv6 CARE tient dans le fait qu'on peut l'utiliser pour tester un programme sans même disposer du code source. La version 3 d'IPv6 CARE permettra de corriger le fonctionnement d'un programme à la volée pour le rendre compatible IPv6.

L'ensemble de ces résultats ont permis au code source de gLite de passer d'une compatibilité de 67% au début du projet à 87% au 30 sept. 2009, et ont été salués par les rapporteurs de la commission européenne pour le projet EGEE. L'ensemble des outils et études réalisés sont utilisables dans un contexte général, particulièrement pour la communauté recherche, qui bénéficie d'une bonne fourniture d'IPv6. Nous espérons donc que cette expérience sera bénéfique à d'autres projets. Ces travaux ont été réalisés avec l'aide et le soutien du G6 [2].

Mots clefs

IPv6, dual-stack, middleware, grille, socket, NAT-PT.

1 Introduction

La grille **Enabling Grids for E-sciencE** (EGEE) est gérée par un middleware gLite qui constitue un véritable système d'information de la grille (interconnexion, authentification et autorisation, acquisition, stockage, gestion du déplacement des données et des jobs). La croissance continue d'EGEE (une cinquantaine de pays connectés), le besoin d'interconnexion aux autres grilles et les recommandations de la commission européenne ont poussé le projet à s'intéresser à IPv6. De plus, les sites de la grille ne peuvent se déployer derrière un système NAT (Network Address Translation). Le but de cet article est d'étudier concrètement comment migrer un tel ensemble de logiciels, écrits dans différents langages, en prenant en compte à la fois leur évolution permanente et leur interaction, et sachant qu'il est impossible de migrer tout le middleware d'un coup.

Dans un premier temps nous rappellerons quelques points techniques concernant l'interopérabilité IPv4/IPv6. Nous expliciterons ensuite les étapes nécessaires à la migration d'un système de cette envergure. Nous montrerons également les méthodes et outils employés pour tester la compatibilité IPv6 (en particulier IPv6 CARE qui vérifie la compatibilité d'un programme au cours de son exécution), ainsi que le travail effectué concernant la programmation IPv6 (en C/C++, Java, Python, Perl). Enfin nous conclurons sur les résultats obtenus dans le projet EGEE. Ces travaux ont été réalisés avec l'aide et le soutien du G6 (http://www.g6.asso.fr/index.php).

2 Rappels concernant IPv6

De l'étude de gLite et de ses dépendances, il est apparu très vite que des modules compatibles seulement IPv4 devaient pouvoir interagir avec des modules IPv6.

2.1 Méthodes de cohabitation entre IPv6 et IPv4

Nous discutons ici les méthodes permettant l'interaction des modules de versions IP différentes (IPv4/IPv6). Nous ne parlerons donc pas des méthodes à base de tunnel, qui visent plutôt à résoudre les problèmes de connexion d'un site à l'Internet IPv6.

Il existe plusieurs types d'approches pour faire cohabiter un environnement IPv6 et un environnement IPv4 :

- La conversion (Translation)
  - au niveau du protocole avec NAT-PT afin de convertir un trafic IPv4 en trafic IPv6 et inversement
  - au niveau de l'application en implémentant une conversion via un Application Layer Gateway (ALG)
- Le « dual-stack »
  - il est possible de configurer une machine en même temps en IPv6 et en IPv4. Dans ce cas les services réseau compatibles IPv6 utiliseront IPv6, les autres utiliseront IPv4.
  - Il est à noter qu'un logiciel serveur sur un système dual-stack peut être implémenté de deux façons : soit le serveur ouvre 2 sockets (une IPv4 et une IPv6), soit le serveur n'ouvre qu'une socket IPv6 (Voir paragraphe 2.2 pour plus de détail).

Pour le cas de gLite, l'approche choisie est le « dual-stack » car cette technologie permet de migrer les services progressivement.

2.2 Méthodes d'implémentation d'un serveur TCP compatible IPv4 et IPv6

Les services gLite utilisent très majoritairement (voire exclusivement) TCP. Sur une machine dual-stack, il existe deux méthodes pour implémenter un serveur TCP capable d'accepter des clients aussi bien en IPv4 qu'en IPv6 :

- Ouvrir deux sockets réseau, une socket IPv4 acceptant uniquement le trafic IPv4, et une socket IPv6 acceptant uniquement le trafic IPv6 (Voir Figure 1)

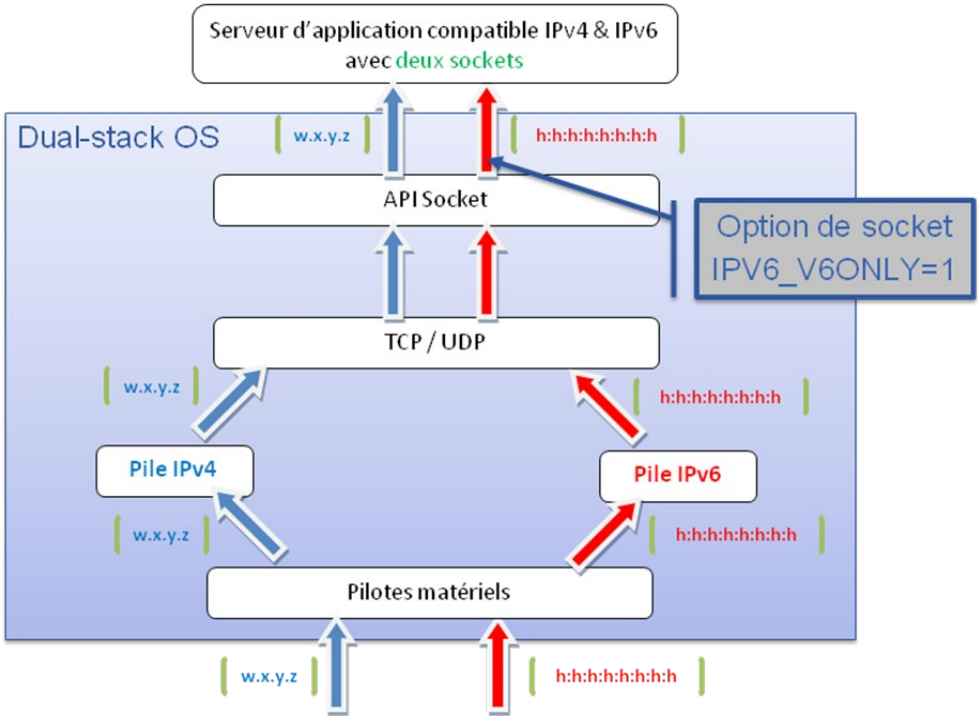

Figure 1 - *Serveur TCP ouvrant une socket IPv4 et une socket IPv6*

- Ouvrir une seule socket IPv6 et la configurer pour qu'elle accepte à la fois le trafic IPv6 et le trafic IPv4 (Voir Figure 2). Dans ce cas le trafic IPv4 passe de la pile IPv4 à la pile IPv6 en étant mappé dans IPv6. Il s'agit d'une technique de conversion de protocole interne à la pile IP.

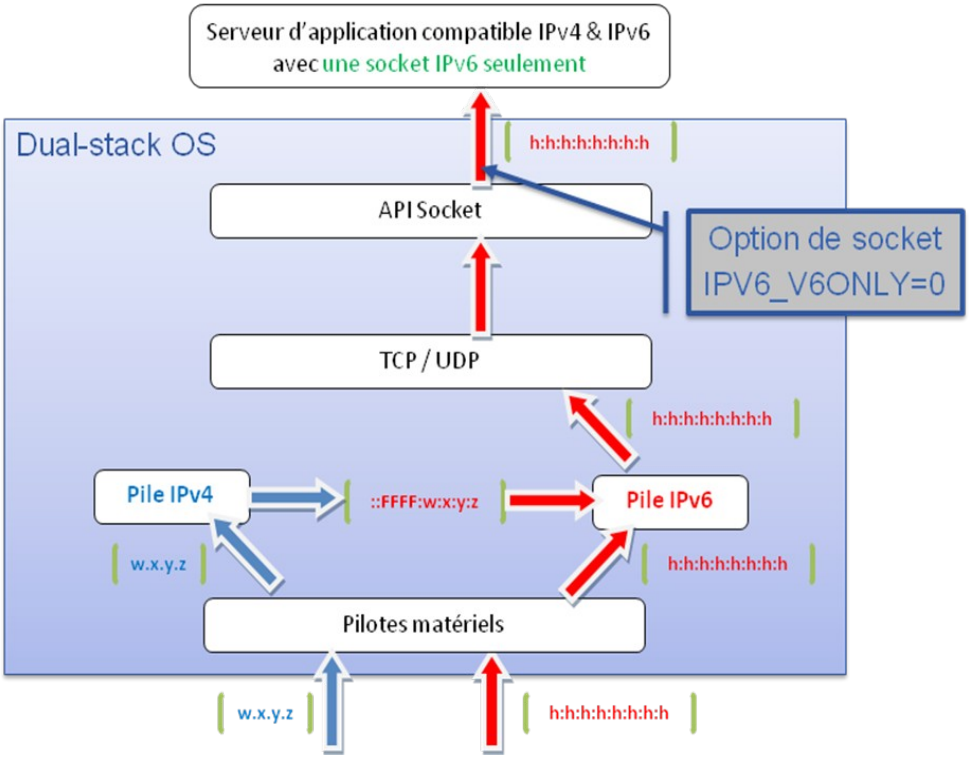

Figure 2 - *Serveur TCP ouvrant une seule socket IPv6*

Le choix entre ces deux fonctionnements doit s'effectuer au niveau du programme, si le langage de programmation le permet, en positionnant l'option de socket IPv6_V6ONLY en conséquence. Si ce n'est pas fait, l'option aura la valeur par défaut définie au niveau du système d'exploitation (fichier **/proc/sys/net/IPv6/bindv6only** dans le cas de Linux), ce qui veut dire que le programme peut ne pas fonctionner sur les systèmes où cette valeur est différente.
Note : Aucune de ces deux méthodes n'a été préconisée dans le cadre de ce projet car, dans les faits, ces deux méthodes offrent le même niveau de fonctionnalité. On notera de plus que les systèmes Linux définissent par défaut les sockets avec l'option IPV6_V6ONLY à 0, ce qui revient à promouvoir la deuxième méthode, quand les systèmes de type BSD la définissent à 1, ce qui revient au contraire à promouvoir la première méthode. Ceci montre que même au niveau des systèmes d'exploitation aucune des deux méthodes n'a réellement pris l'ascendant sur l'autre. Le choix demeure donc celui du développeur, si le langage de programmation offre ce choix.
Pour plus d'information voir [3], [4].

3 gLite : un exemple de Système d'Information complexe

3.1 Présentation de gLite

La grille du projet européen EGEE (60 millions d'euros/an) est utilisée pour fournir des ressources informatiques importantes (principalement en puissance de calcul et espace de stockage) aux projets de recherche scientifique, dans divers domaines (Physique des hautes énergies, Biologie, etc.) et notamment pour le traitement des données extraites du LHC (Large Hadron Collider, le nouvel accélérateur de particules installé au CERN), qui à elles seules représentent 16 Péta octets par an.
Le développement du middleware de grille EGEE, gLite, est industrialisé : les logiciels sont empaquetés (packagés) par un système dédié (ETICS), validés par une équipe spécialisée, testés en pré-production puis déployés sur la grille de production. Le middleware se compose de différents types de nœuds, chacun étant responsable d'une tâche donnée. Par exemple l'un des nœuds gère le système d'information (inventaire des ressources disponibles), un autre gère un espace de stockage, un autre est dédié au calcul, etc.

3.2 Démarche de migration vers IPv6

La difficulté de notre projet était de trouver une démarche pour permettre une migration progressive vers IPv6 qui puisse être compatible avec ces fortes contraintes.
A partir des résultats d'une analyse de la disponibilité d'IPv6 chez les NRENs, qui a montré qu'IPv6 est très largement déployé, nous avons mis en place des bancs de test avec lesquels les développeurs pourraient tester leur application. Ces bancs de test étaient munis de la fonction NAT-PT pour faciliter l'interaction entre les modules de différentes versions d'IP. Parallèlement, nous avons réalisé des outils pour tester la compatibilité d'IPv6 : code checker, IPv6 CARE. Les « bugs » ainsi détectés ont été reportés dans le système de suivi de bug de gLite afin de sensibiliser les développeurs et afin d'avoir une vision de l'évolution de la transition. Nous avons par ailleurs réalisé des études pour faciliter le travail des développeurs. Ces études ont fourni des résultats originaux qui ont également fortement sensibilisé les développeurs. Les dépendances externes de l'ensemble des

logiciels utilisés par gLite ont aussi été analysées. Pour convaincre définitivement les développeurs de gLite, nous avons porté un premier composant à titre d'exemple. Finalement un travail de dissémination s'est aussi révélé indispensable. Celui-ci a été effectué au cours des conférences EGEE, par des présentations et cours à destination des développeurs ou de l'équipe de validation, des posters et des démonstrations.
A l'issue de ce travail plusieurs composants gLite sont compatibles IPv6, et de nombreuses équipes ont placé le portage d'IPv6 sur leur plan de travail pour la fin du projet EGEE-III. La compatibilité IPv6 de gLite est ainsi passée de 67% au début du projet à 87% le 30 septembre 2009.

4 Comment tester la compatibilité IPv6

4.1 En vérifiant les sockets

Le test le plus simple, pour évaluer la compatibilité d'un programme utilisant les sockets réseau, est de vérifier quels sont les sockets qu'il ouvre. La commande netstat est disponible pour cela sur les systèmes du type POSIX.

Pour un serveur, il faut utiliser l'option « -l » pour lister les sockets serveur.

```
[root@quarks IPv6_test]$ netstat -lnpt | grep 2000
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address Foreign Address  State  PID/Program name
tcp        0      0 0.0.0.0:2000  0.0.0.0:*        LISTEN 32343/server_two_so
tcp        0      0 ::2000        ::*              LISTEN 32343/server_two_so
[root@quarks IPv6_test]$
```

Figure 3 - *Vérification des sockets ouvertes par un serveur TCP*

Pour un client, il ne faut pas utiliser l'option « -l », de façon à lister cette fois-ci les sockets connectées (voir Figure 4).

```
[root@quarks IPv6_test]$ netstat -npt | grep 2001
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address         Foreign Address       State       PID/Program name
tcp        0      0 2001:66:32:7::2:2001  2001:66:32:7::3:54104 ESTABLISHED 8046/server_one_so
tcp        0      0 2001:66:32:7::3:54104 2001:66:32:7::2:2001  ESTABLISHED 8047/client
[root@quarks IPv6_test]$
```

Figure 4 - *Vérification des sockets ouvertes par un client TCP*

Une fois que l'on a vérifié que le client se connecte correctement via IPv6, on devra désactiver l'IPv6 sur la machine du client, et vérifier par la même commande que le client est également capable d'utiliser IPv4 pour se connecter.
Note : si le client reste connecté un trop court instant au serveur, cette opération peut être difficile voire impossible à faire. Dans ce cas on devra utiliser une des méthodes qui suivent.

Pour plus d'information sur ces méthodes de test, se référer au document [5].

4.2 En utilisant les logs de firewall

Pour vérifier que la connexion entre un client et un serveur se fait en utilisant IPv6, ou bien IPv4, on peut utiliser les logs des firewalls.
La Figure 5 est un exemple de configuration de ip6tables qui active des logs vers le fichier **/var/log/messages**, pour chaque paquet traversant le firewall, préfixés par **[IPv6 in]** ou **[IPv6 out]** suivant les cas.

```
[root@quarks ~]# ip6tables -F
[root@quarks ~]# ip6tables -X
[root@quarks ~]# ip6tables -N LOG_ACCEPT_IN
[root@quarks ~]# ip6tables -A LOG_ACCEPT_IN -j LOG --log-level 6 --log-prefix "[ipv6 in]"
[root@quarks ~]# ip6tables -A LOG_ACCEPT_IN -j ACCEPT
[root@quarks ~]# ip6tables -N LOG_ACCEPT_OUT
[root@quarks ~]# ip6tables -A LOG_ACCEPT_OUT -j LOG --log-level 6 --log-prefix "[ipv6 out]"
[root@quarks ~]# ip6tables -A LOG_ACCEPT_OUT -j ACCEPT
[root@quarks ~]# ip6tables -A INPUT -j LOG_ACCEPT_IN
[root@quarks ~]# ip6tables -A OUTPUT -j LOG_ACCEPT_OUT
[root@quarks ~]# ip6tables -P INPUT DROP
[root@quarks ~]# ip6tables -P OUTPUT DROP
[root@quarks ~]# ip6tables -P FORWARD DROP
```

Figure 5 - *Configuration de logs pour ip6tables*

Notes importantes :

– Pour réellement utiliser la fonctionnalité standard du firewall, cette configuration est à adapter, car dans l'état elle laisse passer tous les paquets.

– En cas de configuration permanente, il est recommandé d'utiliser un système de rotation de logs paramétré à une fréquence adaptée (toutes les heures par exemple) car ce genre de configuration va rapidement faire grossir le fichier **/var/log/messages**.

– On peut faire une configuration similaire avec iptables pour le trafic IPv4.

La Figure 6 illustre la lecture des logs ainsi générés.

```
[root@quarks ~]# grep "ipv6 out" /var/log/messages | tail -n 3
Sep 30 14:27:09 quarks kernel: [ipv6 out]IN= OUT=eth0 SRC=2001:0660:3302:7006:0000:0000:0000:0008
DST=2001:0660:3302:7000:021d:09ff:fede:dd9b LEN=120 TC=0 HOPLIMIT=64 FLOWLBL=0 PROTO=TCP SPT=22
DPT=45074 WINDOW=3012 RES=0x00 ACK PSH URGP=0
Sep 30 14:27:09 quarks kernel: [ipv6 out]IN= OUT=eth0 SRC=2001:0660:3302:7006:0000:0000:0000:0008
DST=2001:0660:3302:7000:021d:09ff:fede:dd9b LEN=120 TC=0 HOPLIMIT=64 FLOWLBL=0 PROTO=TCP SPT=22
DPT=45074 WINDOW=3012 RES=0x00 ACK PSH URGP=0
Sep 30 14:27:09 quarks kernel: [ipv6 out]IN= OUT=eth0 SRC=2001:0660:3302:7006:0000:0000:0000:0008
DST=2001:0660:3302:7000:021d:09ff:fede:dd9b LEN=104 TC=0 HOPLIMIT=64 FLOWLBL=0 PROTO=TCP SPT=22
DPT=45074 WINDOW=3012 RES=0x00 ACK PSH URGP=0
[root@quarks ~]#
```

Figure 6 - *Lecture de logs de firewall*

4.3 En utilisant un outil de vérification de code source

L'équipe SA2 (Support Réseau) d'EGEE a développé un outil de vérification de code source. Cet outil recherche dans le code source des éléments non compatibles IPv6 (appels de fonctions ou utilisation de structures spécifiques à IPv4, utilisation d'adresses IPv4 dans le code) et permet ainsi d'estimer la compatibilité IPv6 d'un programme. Il est adapté aux langages de programmation utilisés dans gLite : C/C++, Java, Python, Perl.
On peut trouver cet outil dans la section « IPv6 Compliance Testing » sur la page du wiki de l'équipe SA2 [6].
De plus cet outil a été intégré au portail ETICS pour fournir les résultats de la « métrique IPv6 ». ETICS est utilisé pour fabriquer une version officielle de gLite (compilation, paquetage) en vue de son déploiement, les développeurs sont donc informés directement de la compatibilité de leur module. Plus globalement, ETICS est un projet qui fournit une architecture de déploiement et de test pour gLite.

4.4 En utilisant IPv6 CARE

L'UREC a développé un deuxième outil de vérification de compatibilité IPv6, dénommé IPv6 CARE : IPv6 Compliance Automatic Runtime Experiment.
IPv6 CARE analyse l'exécution d'un programme et fournit un rapport sur les appels de fonctions relatifs au réseau, qui permet d'identifier les problèmes de compatibilité IPv6. On peut donc l'utiliser pour tester un programme sans même disposer du code source.

Voici un exemple d'utilisation d'IPv6 CARE, pour évaluer la compatibilité du programme « telnet » (voir Figure 7).

```
etienne@g1:~$ ipv6_care check -v telnet localhost 7777
IPV6 CARE detected: getaddrinfo() with [ ai_family=AF_UNSPEC ai_socktype=SOCK_STREAM nodename=localhost
servname=7777 ]
IPV6 CARE detected: getnameinfo() with [ sa.ip=127.0.0.1 sa.port=7777 ]
Trying 127.0.0.1...
IPV6 CARE detected: socket() with [ domain=AF_INET type=SOCK_STREAM protocol=ip ]
IPV6 CARE detected: connect() with [ socket=3 address.ip=127.0.0.1 address.port=7777 ]
IPV6 CARE detected: getnameinfo() with [ sa.ip=::1 sa.port=7777 ]
Trying ::1...
IPV6 CARE detected: close() with [ fd=3 ]
IPV6 CARE detected: socket() with [ domain=AF_INET6 type=SOCK_STREAM protocol=ip ]
IPV6 CARE detected: connect() with [ socket=3 address.ip=::1 address.port=7777 ]
telnet: Unable to connect to remote host: Connection refused
IPV6 CARE detected: close() with [ fd=3 ]
-----------------------------------------------------------------------------------------------------------
IPv6 diagnosis for 'telnet localhost 7777' was generated in:/tmp/ipv6_diagnosis/telnet/by_pid/pid_32756
-----------------------------------------------------------------------------------------------------------
etienne@g1:~$
```

Figure 7 - *Utilisation d'IPv6 CARE (1)*

On peut facilement observer que telnet a un comportement conforme à IPv6.

Essayons maintenant de diagnostiquer la compatibilité IPv6 du serveur mysql (Figure 8) :

```
root@g1:~# ipv6_care check -v /usr/sbin/mysqld
IPV6 CARE detected: inet_addr() with [ cp=127.0.0.1 ]
091006 16:35:29  InnoDB: Started; log sequence number 0 43655
IPV6 CARE detected: socket() with [ domain=AF_INET type=SOCK_STREAM protocol=ip ]
IPV6 CARE detected: bind() with [ socket=10 address.ip=127.0.0.1 address.port=3306 ]
IPV6 CARE detected: listen() with [ socket=10 backlog=50 ]
091006 16:35:29 [Note] /usr/sbin/mysqld: ready for connections.
Version: '5.0.75-0ubuntu10.2'  socket: '/var/run/mysqld/mysqld.sock'  port: 3306  (Ubuntu)
[...]
091006 16:35:49 [Note] /usr/sbin/mysqld: Normal shutdown
IPV6 CARE detected: close() with [ fd=10 ]
091006 16:35:49  InnoDB: Starting shutdown...
091006 16:35:50  InnoDB: Shutdown completed; log sequence number 0 43655
091006 16:35:50 [Note] /usr/sbin/mysqld: Shutdown complete
-----------------------------------------------------------------------------------------------------------
IPv6 diagnosis for '/usr/sbin/mysqld' was generated in: /tmp/ipv6_diagnosis/mysqld/by_pid/pid_19367
-----------------------------------------------------------------------------------------------------------
root@g1:~# cd /tmp/ipv6_diagnosis/mysqld/by_pid/pid_19367
root@g1:pid_19367# ls
full_command_line  log  possible_ipv6_related_errors
root@g1:pid_19367# ls possible_ipv6_related_errors
inet_addr
root@g1:pid_19367# cat possible_ipv6_related_errors/inet_addr/problem_description
PROBLEM DETECTED:
A call to inet_addr() which is IPv4-only was detected.
SOLUTION:
You should use getnameinfo() with the flag NI_NUMERICHOST, in order to be address-family agnostic.
root@g1:pid_19367# cat possible_ipv6_related_errors/inet_addr/process_stacks
One call was done at 16h35mn28s. Process function calls stack was:
Function:                                                    | Source file name and line number:
-----------------------------------------------------------------------------------------------
flush_thread_cache()                                         | *
handle_options                                               | *
kill_server_thread                                           | *
main                                                         | *
*: code file and line number unavailable (source needs to be compiled with option -g)
root@g1:pid_19367#
```

Figure 8 - *Utilisation d'IPv6 CARE (2)*

Comme on le voit dans ce deuxième exemple, on peut consulter le détail de l'analyse dans le répertoire généré par l'outil : ici **`/tmp/IPv6_diagnosis/telnet/by_pid/pid_19367`**. Dans ce répertoire, on trouve notamment des informations pour chaque appel non compatible IPv6 détecté : sa localisation dans le code, ainsi qu'une indication sur la façon de modifier le code pour le rendre compatible IPv6.
L'outil IPv6 CARE ainsi que sa documentation est disponible à l'adresse [7]. La nouvelle version 3 d'IPv6 CARE visera à corriger à la volée les appels aux fonctions réseau effectués par le programme cible, afin de rendre celui-ci compatible IPv6 sans modification.

4.5 En utilisant un banc de test IPv6

L'équipe de travail sur IPv6 de SA2 a également construit un banc de test composé de deux sites de grilles, l'un à Paris, l'autre à Rome (Voir Figure 9). Les deux sites sont reliés par les réseaux RAP, RENATER, GEANT et GARR, qui fournissent tous une connectivité IPv6. Chaque site est composé des principaux types de nœud de la grille.

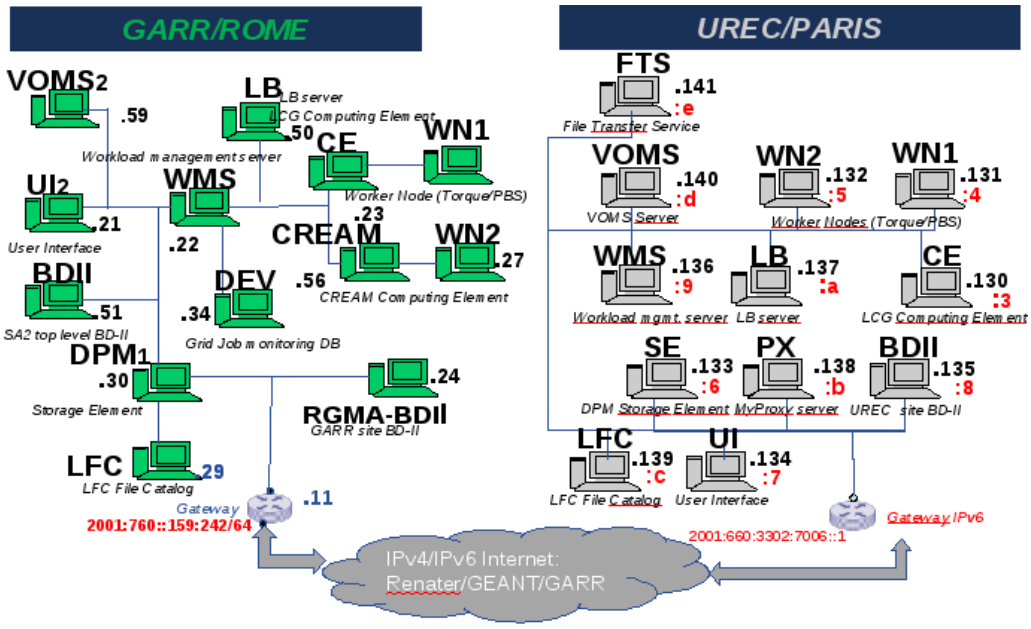

Figure 9 - *Le banc de test IPv6 d'EGEE*

Ce banc de test est utilisé par les développeurs car ils n'ont parfois pas d'environnement IPv6 disponible pour tester leurs composants. Il est également utilisé pour démontrer et valider de manière globale les fonctionnalités IPv6 de la grille.

5 Comment rendre les programmes compatibles IPv6

5.1 En utilisant l'API IPv6

La compatibilité IPv6 d'un programme dépend en grande partie de la compatibilité offerte par le langage de programmation en lui-même. Nous avons donc réalisé une étude des fonctionnalités IPv6 des langages C/C++, Java, Python et Perl, qui sont utilisés dans gLite (voir [8]). A partir d'exemples simples dans chacun de ces langages (un client et un serveur TCP capables de communiquer dans des environnements IPv4 et/ou IPv6), l'étude montre que migrer un code est une opération aisée. Par ailleurs, les difficultés et la quantité de travail globale requise pour une migration y sont aussi détaillées :
- Nécessité d'avoir accès au système dans le cas de Java (vérification/modification d'un paramètre système) et Perl (installation d'une librairie). Ceci n'est pas le cas pour C/C++ et Python.
- Nécessité de modifications de code source plus importantes dans le cas de Python (si utilisation de l'API bas niveau), Perl (si utilisation de l'API bas niveau) et C/C++ que pour Java.
Enfin cette étude met en valeur les avantages d'une programmation de haut-niveau (en utilisant une librairie compatible IPv6 chargée de gérer les appels réseaux, ou en créant des services basés sur xinetd, voir paragraphe 5.2 ci-après).
Se référer au document [8] pour plus d'informations.

5.2 En utilisant Xinetd

Le « super-démon » xinetd est un programme qui permet de factoriser la partie gestion réseau d'un certain nombre de services (telnetd, ftpd,...). L'implémentation d'un service réseau basé sur xinetd se limite alors à la procédure de gestion d'une seule connexion client, sans avoir à utiliser l'API sockets, car xinetd s'en charge. Outre la simplification que cela engendre, on notera que xinetd étant compatible IPv6, les services basés sur xinetd le sont aussi. Il suffit d'indiquer « flags = IPv6 » dans le fichier de configuration du service pour que xinetd ouvre une socket IPv6.
Par conséquent une méthode possible pour rendre un service compatible IPv6 est de l'adapter pour l'utiliser via xinetd. (Cela se limitera cependant à des services avec un fonctionnement classique et ne nécessitant pas d'optimisation poussée concernant le nombre de clients simultanés à gérer).

5.3 En utilisant IPv6 CARE

Une nouvelle version de IPv6 CARE (version 3.0) est en cours de développement au moment de la rédaction de cet article. Cette version proposera deux modes :

−Le mode « checking » qui fournit le diagnostique comme actuellement (voir paragraphe 4.4).

−Un nouveau mode « patching » qui permettra de modifier le fonctionnement du programme à la volée pour le rendre compatible IPv6.

L'utilisation du mode « patching » est particulièrement intéressante pour les exécutables ou librairies d'exécutables dont on ne maîtrise pas le développement. Dans le cas du middleware gLite par exemple, une liste de ses dépendances externes a été établie et certaines d'entre elles se sont révélées non-compatibles IPv6. Par exemple mysql est utilisé par certains noeuds gLite et sa non-compatibilité IPv6 pose problème. Ce nouveau mode d'utilisation d'IPv6 CARE pourrait être une solution.

6 Conclusion et principaux résultats obtenus

Le principal résultat obtenu de ce projet de migration de gLite vers IPv6 est le fait d'avoir démontré que gLite peut tourner sur une architecture dual-stack. A partir de ce résultat, la voie est ouverte pour une migration progressive de gLite. En effet, on peut migrer un à un les nœuds en faisant cohabiter les composants non encore migrés (utilisant IPv4) et les composants déjà migrés (utilisant de préférence IPv6).
De plus, certains services sont déjà compatibles. A l'heure actuelle il s'agit du système d'information (BDII), de l'élément de stockage (DPM/LFC), de l'élément dédié au calcul (CREAM), de l'élément dédié à la gestion des jobs (WMS), et de la librairie utilisée pour le transfert de fichiers (GridFTP). D'autres nœuds sont en cours de migration et/ou leur migration est intégrée au planning de l'équipe de développement, ce qui montre qu'à l'issue d'EGEE-III une partie importante des nœuds seront compatibles IPv6. Globalement la compatibilité du code de gLite est ainsi passée de 67% au début du projet pour atteindre 87% le 30 septembre 2009. Cette compatibilité sera un atout pour la pérennité de gLite.
Les résultats de l'équipe sont excellents compte tenu des restrictions budgétaires dont a souffert le projet. Lors de la première revue du projet EGEE III, les rapporteurs de l'union européenne ont ainsi mis en lumière les progrès accomplis dans cette partie du projet.
« *The reviewers would also like to highlight a number of areas in which the project has made particular progress compared with the status at the end of EGEE-II:*
- *[…]*
- *Significant progress in IPv6 compliance, with numerous middleware components adapted and the dual-stack operation of gLite being demonstrated in April 2009.*
- *[…]* »
Enfin, les outils développés lors du projet se sont montrés très utiles, et leur emploi n'étant pas spécifique au système gLite, ils pourront être utilisés dans d'autres projets.

Bibliographie / Références

[1] Site web du projet EGEE, http://www.eu-egee.org/
[2] Site web du G6, http://www.g6.asso.fr/
[3] W. Richard Stevens, Bill Fenner, Andrew M. Rudoff, *UNIX Network Programming*, Addison Wesley, Février 2007
[4] Gisèle Cizault, *IPv6 Théorie et Pratique*, http://livre.g6.asso.fr, Novembre 2005
[5] Etienne Dublé, *How to test IPv6 compliance of a socket server*, https://edms.cern.ch/document/930868, Octobre 2008
[6] Page wiki de l'équipe IPv6 de EGEE SA2 : https://twiki.cern.ch/twiki/bin/view/EGEE/IPv6FollowUp
[7] Site web de IPv6 CARE : http://sourceforge.net/projects/IPv6-care
[8] Etienne Dublé, *IPv6 Compliance With C/C++, Java, Python and Perl*, https://edms.cern.ch/document/971407, Novembre 2008