RESEARCH PAPER

Traffic engineering and Quality of Services allowing to interconnect SAN applications

Département d'informatique
Université du Québec À Montréal
Case Postale 8888, succursale Centre ville
Montréal (Québec) Canada – H3C 3P8

Abstract

For applications using SAN interconnections (AuSIs) Quality of Service (QoS) in data transferring is fundamental. First AuSI needs the flexibility to control the establishment of connexions between themselves and logical units (LUNs).Second AuSI has to be sure that the QoS is respected during the data transferring. Signalling and control approaches using Web Services have been proposed to provide SAN interconnections services. The difficulty is that components managing which participate to the SAN interconnections architecture do not depend on our program. This underlines the network resources providers management by themselves and the remote SAN architecture management by themselves too. Hence, our signalling solution takes account of the managements constraints imposed by the different components.

Not only this paper presents a web services based approach to provide SAN interconnection services and it also presents algorithms to manage network resources. The network resources are given by a provider and our program have to gather requests of applications into with a goal of to avoid abusive network resources ordering. We present an architecture based on web services allowing applications to see and choose SAN interconnection services. Then we present the core of our system allowing to ensure the respect of the QoS for each applications and allowing to manage network resources. We present the integration of an admission control module and the integration of a traffic engineering module. We detail the experimentation results concerning both modules described above to show the viability of the approach.

Keywords : SAN interconnection services, respect of QoS, network resources management, avoid abusive network resources ordering, web services approach, resource reservation signalling.

1. Introduction

For applications using SAN interconnections (AuSIs), Quality of Service (QoS) in data transferring is fundamental. First AuSI needs the flexibility to control the establishment of connections between themselves and logical units (LUNs). Second AuSI has to be sure that the QoS is respected during the data transferring.

This has led to a new network paradigm called applications semi controlled network. On one hand Applications-controlled network allows each application to order the network resources according to their need. On the other hand a program controls the requests of AuSIs to provide a Quality of Service and also to manage the network resources.

The goal of this project is to give the ability for AuSIs to dynamically apply for network resources. But the goal of this project is also that our program manages the network resources. New signalling approach using Web Services has been developed to realize that project. The applications-semi-controlled network allows AuSI to choose a path to communicate with a LUN. This path is composed of several network roads due to the heterogeneous environment that the path crosses. The AuSIs choose freely a path among those that the project built.

Due to the fact that protocols used to realize SAN interconnections are very recent there is no anterior projects about traffic engineering and QoS allowing for AuSIs. Anyway the project proposes a solution to resolve the need of QoS for AuSI but also to manage network resources that a provider can allow.

The key idea to ensure that the QoS will be respected is to introduce an admission control mechanism to regulate the network traffic. Moreover to ensure that the network resources will be managed as best as possible, the second idea is to introduce an Engineering mechanism like for example an algorithm based on Hopfield network. As a result, the AuSIs can control the network resources that they need but it is another program which manages the network resources.

Our solution is based on Web Services and introduces admission control and engineering mechanisms. This work demonstrates that complex engineering control is required for proper functioning and management of network resources. We have implemented the approach and we conducted experiments on a virtual network due to the fact that we did not possess the good equipments to really test the program. However, the experimentation results show the viability of the approach. Then we will present some details about this results.

The remainder of this paper is organized as follows. Section 2 describes the design approaches and the SAN interconnection system architecture. Section 3 explains how SAN interconnections are realized and how the network resources are managed by the program. In section 4, we describe our implementation. In section 5, we present the experimentation results. Finally, concluding remarks are given in section 6.

2. SAN interconnection network layer model

This section presents the architecture used to develop the project. The knowledge of the network which allows the SAN interconnections is fundamental for our program because it have to offer different paths which come from an local SAN iSCSI router to a remote SAN LUN.

2.1 SAN interconnection architecture

This section describes the model our program used to deploy SAN interconnection services. We based our architecture on the following model. The picture shows the different parts of the architecture. To realize SAN interconnections, our program considered that a path could be divided in three parts :

- the first part of the path representing a link between a local SAN iSCSI router and a local SAN border router
- the second part of the path representing a link between a local SAN border router and a remote SAN border router
- the third part of the path representing a link between a remote SAN border router and a remote SAN LUN (through an iSCSI router)

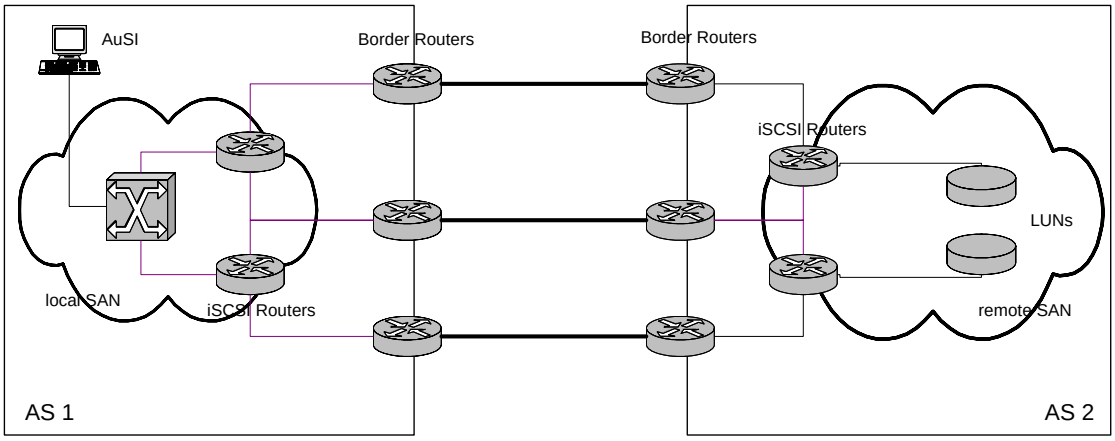

Figure 1 : SAN interconnection overview

The following points describe the design approach our program has set up to carry out SAN interconnections.

1) Maximizing remote SAN autonomy. The only interaction between the local SAN and the remote SAN is based on Web Services. The remote SAN can manage its network topology as it wants. Then to be sure to offer good SAN interconnection services, our program periodically search for the remote SAN topology in order to rebuild good paths.

2) No automatic order for network resources. The only order for network resources is realized when the program try to place some AuSI requests and when the program does not find enough free network bandwidth.

3) Resolving the concurrent AuSI requests. Because new application requests can arrive at any time, we decided to define a periodicity in our program. In this way our program is periodically looking for some new requests. On each "tick" (representing the end of a period), our program treats all the requests it received. If some new requests appear after a period, the program deals with them at the beginning of the next period.

2.2 Architecture overview

To understand with which entities our project will communicate, we present the local SAN architecture.

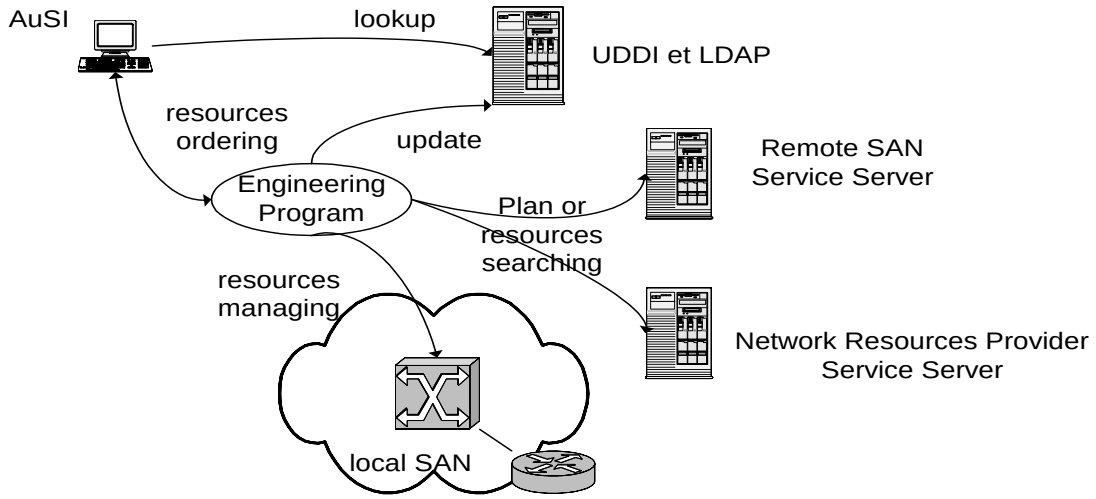

Figure 2 : Communications between the program and the other entities

On this picture we can see several entities introducing all the communications with our program. There are four different parts which resume the program functioning.

- The first communication phase is to establish the SAN interconnection services. To ensure this first part, our program uses web services between our program and remote SAN Service Server and Network Resources Provider Service Server. Then our program set up the services in the UDDI/LDAP

- The second communication phase concerns the AuSIs and the service agent (which deals with the applications). The Service Agent is the interface which provides SAN interconnection services to the applications

- The third communication phase concerns recovery and process of application requests and as needed the communication between the engineering mechanism and the network resources provider

- Finally the fourth part concerns the communication between the program and the UDDI/LDAP entity because of periodical service updates

3. Architecture description

In this section, we describe the different layers which compose the architecture of our program. We detail the service agent mechanism and the application requests processing.

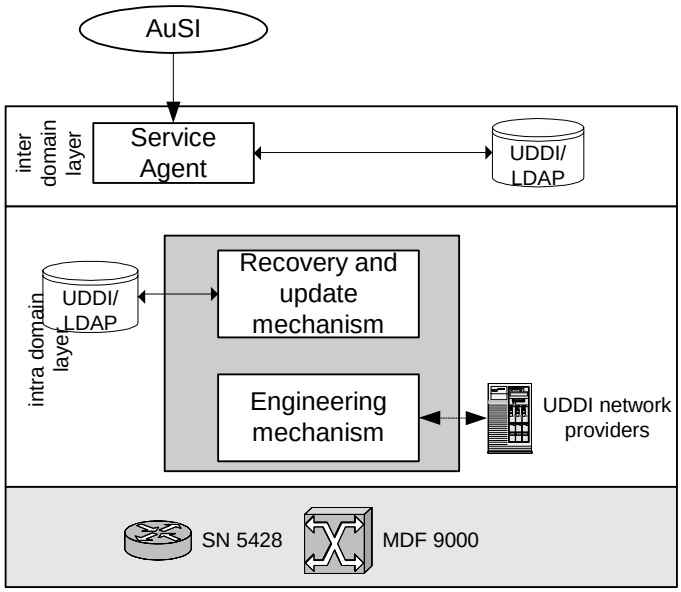

Figure 3 : the program architecture layers

3.1 Inter-domain layer

This part describes the service agent mechanism. The service agent is responsible to present the different SAN interconnection services to the AuSIs. Hence, this module manages the communication processing introduce by AuSIs. It includes the login/logout process and the search and display SAN interconnection services process.

SAN interconnection services searching

In our project, the SAN interconnection services are put in the UDDI/LDAP entity. Then, the service agent have to search the SAN interconnection services to present them.
The service agent have to search the different parts composing a path which represents a VSAN service. Because the recovery and update mechanism builds the path it is easy for the service agent module to present the path. The path

concatenation that the service agent have to do is facilitate by the LDAP organization due to the fact that data recording is based on Common Interface Model (CIMv2) schema.

We present a logical schema which resumes the inter-domain layer :

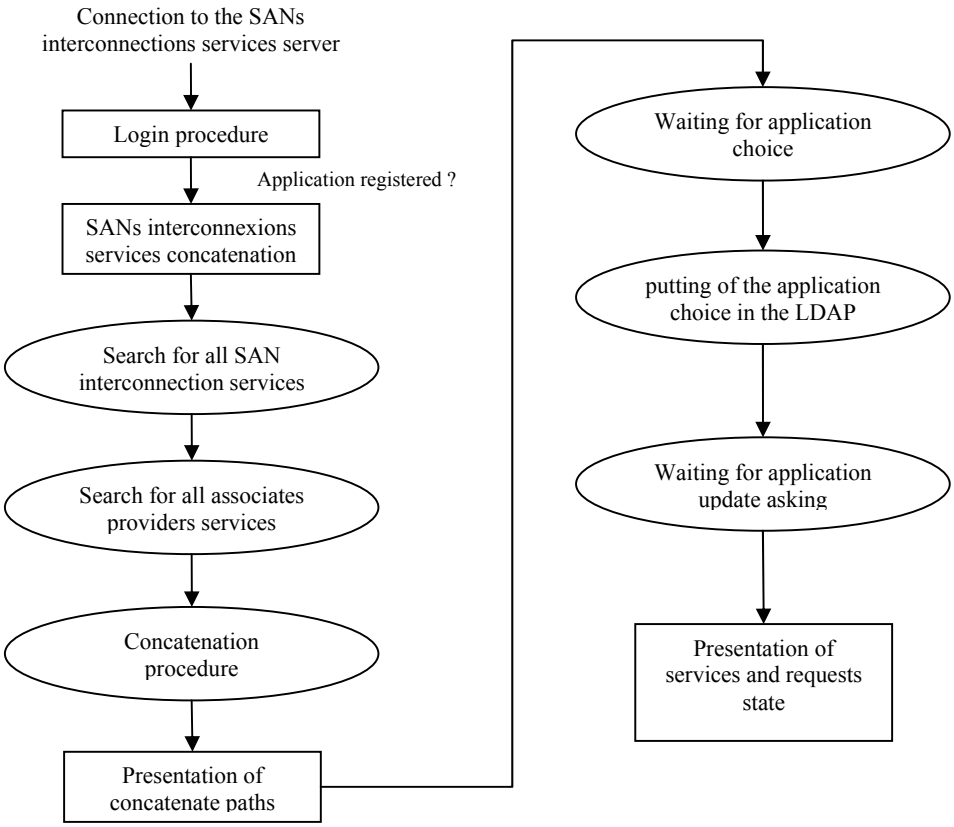

Figure 4 : Service Agent work description

3.2 Intra domain layer

The intra domain layer represents the internal modules which composed our program. This layer is based on three central modules :

- The information collector module
- The traffic engineering module
- The admission control module

To ensure the communications between two of these three modules or between one and another entity (for example LDAP or network resources provider web service server), we based our model on web services architecture. The advantage provided by this technology is the independence between the three modules listed above and between the rest of the world. Indeed, if implementation of network resources providers changes, the three modules don't care because the web services interface which permits the communication doesn't change. Indeed, every module can change but it have to answer the web services interface all the time.

To increase the reliability of our project, we included web services communication even between two modules. Therefore, if one of the three modules listed above have to be re implemented, the other modules can be unchanged.

Concerning the work of the intra domain layer, it have periodically to do some tasks :

- recover new AuSIs requests containing in the LDAP
- recover network information

- call the traffic engineering module if some new AuSI requests appear
- call the admission control module if not
- update the state of
    - AuSI requests
    - VSAN services
    - Provider services

The following diagram represents the work of the intra domain layer :

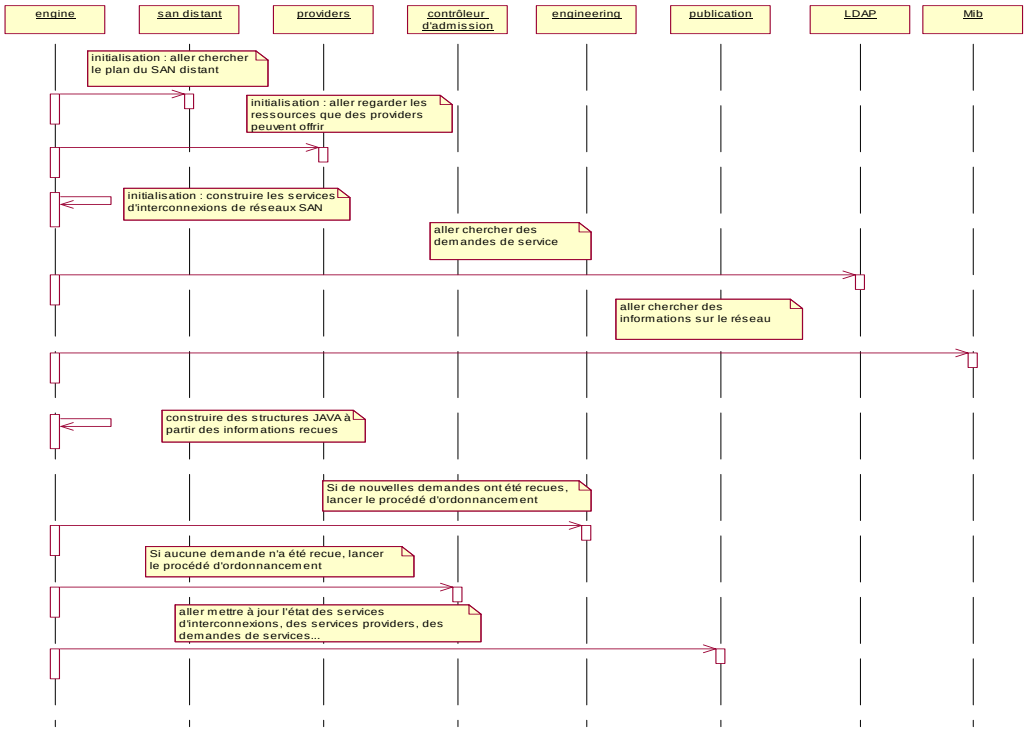

Figure 5 : Intra domain layer work

4. Description of our solution

This section describes the approach of our solution to manage the network resources and to ensure the QoS for all the applications which use our program.

4.1. SANs interconnection services managing

In this section we present the deployment of paths which allow applications to communicate from themselves to a LUN. Therefore we describe the processes to retrieve the architecture of a remote SAN. After that, we describe the processes to build a path from a local iSCSI router to a remote LUN. For that, we describe the algorithm which consist to concatenate the different paths due to the different environments from a local SAN to a remote SAN.

4.1.1 How to obtain SAN interconnection services

To retrieve the environment which compose the remote SAN, our program has to communicate

- with a remote SAN
- with a list of network providers

Communication with a remote SAN :

The communication is only based on Web Services technology. Then our program periodically updates the view of the remote SAN that it has. To realize the communication, we imagined that each SAN published the plan of his network. For example, each SAN can implement a Web Service which gives a description of the SAN architecture.

Our program have to look in a public directory (for example an UDDI) to find the address of the Web Service. After that, our program can bind on the Web Service to recover the SAN architecture.

Communication with a network provider

The communication is also based on Web services technology. Our program contacts a list of network provider that it has. Then it obtains some network services between two border routers. The processes to order network services are the same that before.

Our program has to recover the binding address corresponding to an access point which permit to communicate with the network provider Web Service. After, our program send an purchase order via the Web Service to order some network resources. To finish, our program is waiting for the answer of the network provider. If the network provider allows us the network resources our program needs, it does not contact any other provider.
A figure showing the different steps described above follows.

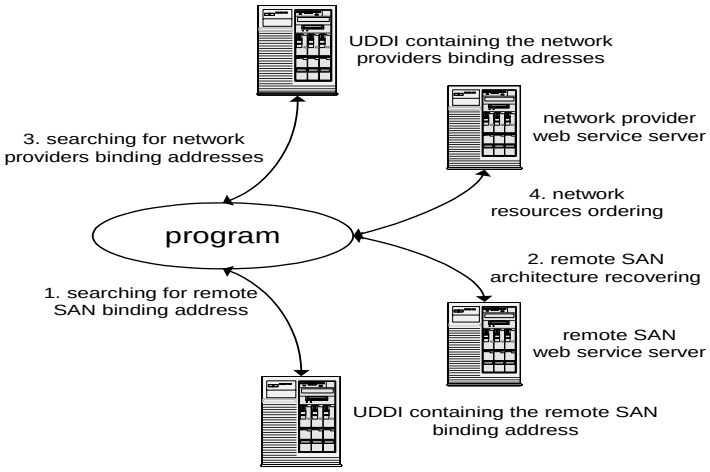

Figure 6 : Web Services communication in SAN interconnections services building

After these preceding steps, our program possesses all the paths corresponding to the different environments listed in page 2. But there is no associations between all these paths.

The next and last step our program needs to do, is the concatenation of all these paths. As showed figure 1, paths between border routers are associated with paths of the local SAN architecture and paths of the remote SAN architecture. To realize the concatenation, our program uses a simple algorithm. To build SAN interconnection services, the algorithm is divided in four parts :

- take a provider service that network provider give us
- build a path which is an association between local SAN iSCSI router and local SAN border router
- build a path which is an association between remote SAN border router, remote SAN iSCSI router and remote SAN LUN
- build an association between the paths building in the second and in the third step

The different figures below show the path construction.

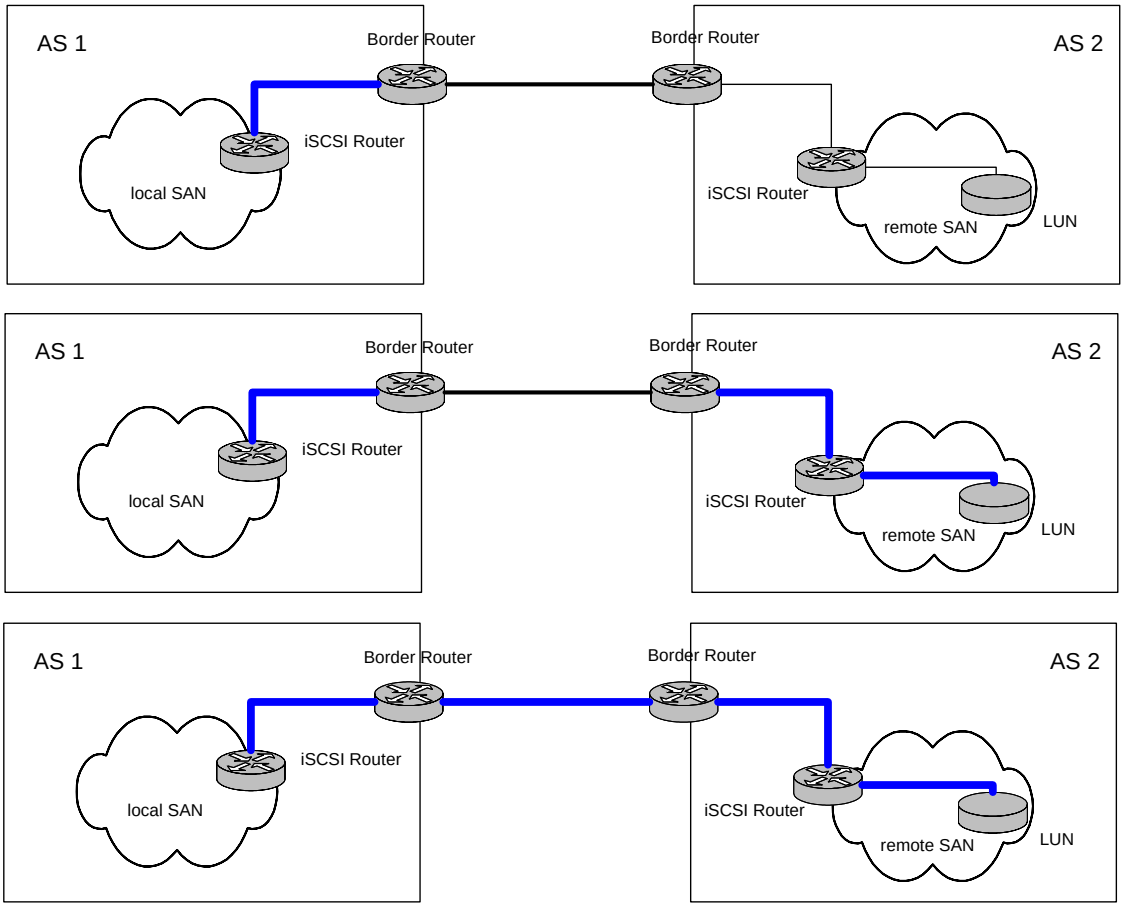

Figure 7 : SAN interconnection service building

The first figure shows that the path representing the SAN interconnection service is represented by only the part of the local SAN. The second figure shows that the path is represented by the local and the remote SAN architectures. It shows that our program associates two paths. Then, the last figure shows the joint that the program have to do to make a provide a SAN interconnection service.

4.1.2 traffic engineering on SAN interconnection services

Our program implements a traffic engineering module which has two specificities :

- to manage the network resources that a provider gave us
- to gather application requests into a provider service in order to avoid abusive network resources ordering

Our program proposes to consider groups of AuSI requests. Indeed the definition we could give to represent an AuSIs group will be the following :

"*an AuSIs group is represented by a set of requests which want to communicate with the same remote LUN*"

Then we also consider groups of SAN interconnection services. In the same way, the definition we could give to represent a group of SAN interconnection services will be the following :

"*a group of SAN interconnection services is represented by a set of SAN interconnection services which permit to communicate from anywhere to the same remote LUN*"

Then, the goal of the algorithm is to find a projection from the first set of groups to the second set of groups as we can see in the following picture :

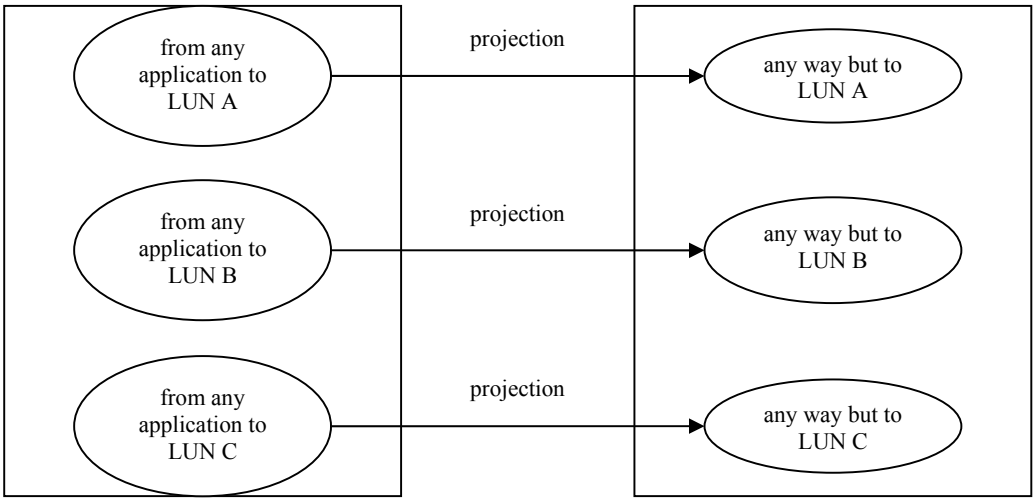

Figure 8 : traffic engineering by translation resolving

To find a translation which manages network resources, we based our program on two different factors :

- time
- capacity

Indeed after to build the different AuSI groups, our program represents them on a two dimensions graph. The first dimension represents the time and the second the capacity. Then the graph has a maximum in terms of requested bandwidth.
The maximum can be represented by the following expression :

$$\beta_{\max} = \sum_{i \in I} capacity(i)$$

where :

- I is a set of AuSI requests
- i is a request of this set

Then our program defines two variables representing the lower time and the higher time which belong to requests taking part in the computation formula to obtain the above $\beta_{\max}$:

- $t_{\max} = \underset{i \in I}{Max}(t_{end}(i))$
- $t_{\min} = \underset{i \in I}{Min}(t_{start}(i))$

where t_{min} represents the starting time of AuSI requests and t_{max} the ending time of AuSI requests

After that, our program looks in the corresponding SAN interconnection services set for two different things :

- First it looks for a SAN interconnection service which could contain all the AuSI requests participating to the maximum. To find a such service the two following conditions have to be respected :
  - $\beta_{service} \geq \beta_{\max}$
  - $t_{service_{start}} \leq t_{\min} \wedge t_{service_{end}} \geq t_{\max}$
- Second, it looks for a set of services which could contain all the AuSI requests. To do that, the two conditions listed above change :
  - $\sum_{j \in J} \beta_{service(j)} \geq \beta_{\max}$
  - $\forall j \in J \wedge \forall k \in K, t_{service(j)_{start}} \leq t_{k_{\min}} \wedge t_{service(j)_{end}} \geq t_{k_{\max}}$
  - Where J represents a group of SAN interconnection services and K a subgroup of I defined by the fact that all requests in K will be put into a service of J.

After that, three possibilities can happen :

- Our program finds service corresponding to the first and second condition. Then it puts all the AuSI requests in it and the program continues the traffic engineering with another AuSI group.
- Our program finds some services which answer to the third and fourth conditions. Then it tries to put all AuSI requests in their with the following conventions
    - the requests which apply for the biggest bandwidth are putting into the same service at first
    - when the biggest request cannot be putting in the same service, the program tries to put another, to maximize the service utilization
    - while some requests are not putting into a service and while there is possibilities to put them into services, the program continues his execution
- our program does not find any service which satisfy conditions or it cannot answer the conventions listed above. Therefore, it has to order network resources to a provider

4.2. SAN interconnection services control

This section describes how the admission control module works. It describes the means used to ensure that the QoS will be respected for all the applications which used our program to establish a communication through SANs. To control that the QoS is respected, the module needs network road information. Therefore our program is based on information recover due to the Management Information Base (MIB) utilization.

We based the QoS respect on the following mathematical expression :

$$N(P_i) \equiv \sum_{requests \in Pi} (bytes + remaining)$$

where

- P_i represents the path (with the number i) between two border routers
- Requests represent the set of AuSI requests using the path P_i
- N(P_i) represents the number of transmitted bytes in the path P_i
- Remaining represents the number of bytes added last time that the program detected that the anterior configuration did not respect the QoS. The remaining is very important because it permits to prevent that the same error occurs in every period.

To understand the above problem we can illustrate the work of the admission control module by the following pictures.

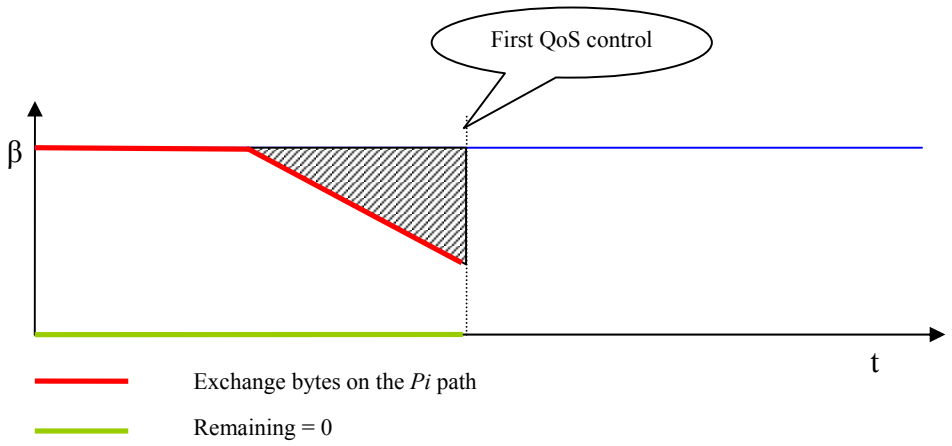

Figure 9 : first admission control test fails

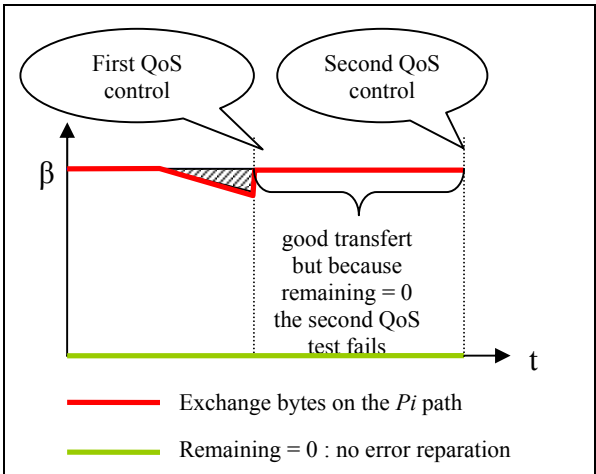

Figure 10 : no error reparation

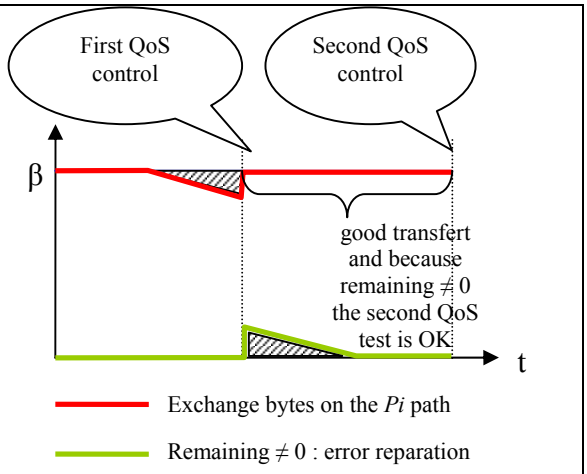

Figure 11 : error reparation

Our program uses the following expression to know the number of bytes which have been exchanged if the QoS was perfectly respected :

$$N_{ideal} = \frac{\beta_{wished}}{t_{actual} - t_{start}}$$

Then our program has just to find the result of the following expression :

$$R = N_{ideal} - N(P_i)$$

Thus if the difference is close to null, the QoS is respected. Otherwise, the program has to find a new configuration to ensure that the QoS will be respected. To realize this, the admission control module call the engineering module with a new parameter representing bad SAN interconnection services in order to avoid that the traffic engineering module give the same configuration than before.

5. Experimentations and Simulations

This section presents the different simulations we have realized. Different experiments were conducted on a small virtual network architecture. The experiments analyse

- the signalling time to recover provider paths and remote SAN architecture
- the execution time of our program to find an engineering solution
- the execution time of our program to verify that QoS is respected and to reschedule AuSI requests
- the execution time of our program to order network resources and to schedule AuSI requests

We have based our tests on the following virtual network :

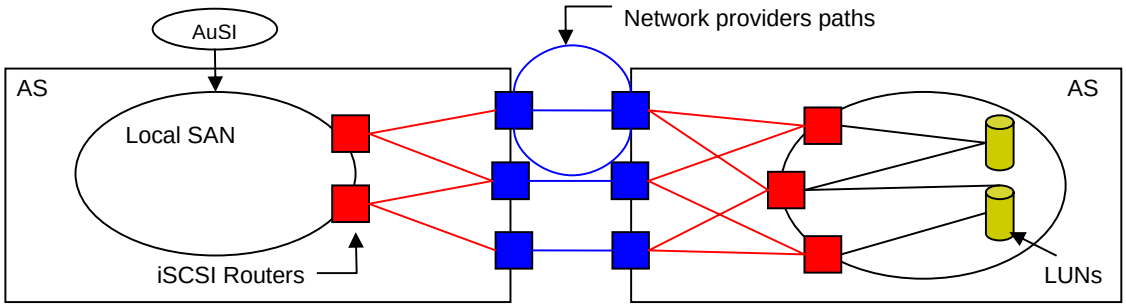

Figure 12 : experimental network

5.1 Signalling time to recover providers paths and remote SAN architecture

Here we present the time to discover the different parts which participate to a SAN interconnection service.
The result is T = time to call provider web service server + time to call remote SAN web service server

Finally T = 1543 + 1642 = 3185 ms

5.2 Engineering solution time

Here we present the time to find a good configuration when no problems appear i.e. our program possesses SAN interconnection services which can support all the AuSI requests our program has to treat. The time contains all the web service calls, the algorithm execution time to find a solution and the time to recover the configuration. We present a graphic representing the approach we have implemented. The execution time depends on satisfies properties listed page 9 i.e. if the second property is respected : our program possesses some services. The free bandwidth sum of these services can treat all the AuSI requests.

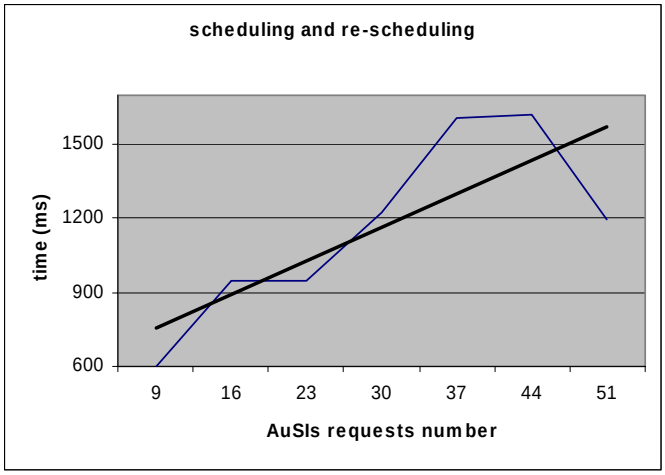

Figure 13 : linear variation

5.3 Admission control execution time

We present the execution time to check that QoS is respected for all the AuSIs. Then, the result depends on the number of SAN interconnection services our program possesses. A graphic displays the time average in function of SAN interconnection services number.

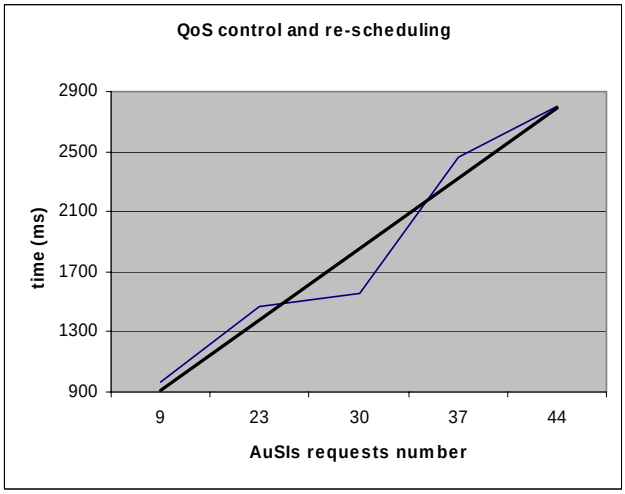

Figure 14 : QoS control linear variation

5.4 Re-engineering execution time after a resources network provider ordering

Because all the communications between modules are based on web services model, our program is highly reliable. Unfortunately each web service call takes a non null time. Then, during a re-engineering phase after to have order new network resources we have measured the execution time. As above, this time depends on AuSI requests number. So we present a graphic displaying the time average.

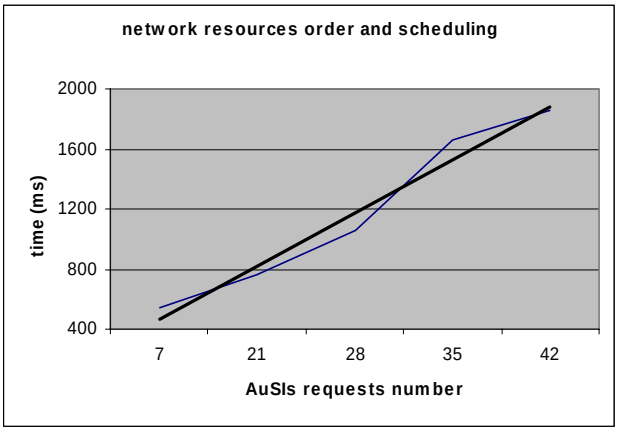

Figure 15 : network resources order constant time

6. Conclusion

In this paper we have presented a web services based approach to provide SAN interconnection services but also algorithms. These algorithms permit to manage network resources that a provider gave us and to gather AuSI requests into a provider service in order to avoid abusive network resources ordering. The architecture allows each AuSI to apply for SAN interconnection services and it also manages the network resources. Moreover, with the proposed signalling, AuSI requests can reserve SAN interconnection services in advance according to scheduling algorithm realized by the traffic engineering module.

Our web services architecture allows remote SAN to change his architecture because of periodically updates of remote SAN architecture. For the same reason, it also allows the possibility for a network provider to join or leave the provider list. Indeed our architecture only needs to find in an UDDI a correspondent provider list to retrieve binding address and to recover the corresponding network resources that a provider can offer. Finally our architecture allows AuSIs to use our program with only the binding address of our service agent module. Only functions present in the service agent interface have to be known by the AuSIs.

The traffic engineering and admission control modules increase the reliability of our approach. Scheduling allows a good management of network resources and admission control allows a necessary quality of service for all the AuSIs.

References

[1] MIB CISCO 5420 SN :
http://www.cisco.com/en/US/products/hw/ps4159/ps2160/products_data_sheet09186a00800910f2.html

[2] MIB CISCO 9000 MF :
http://www.cisco.com/en/US/products/hw/ps4159/ps4358/products_mib_quick_reference_chapter09186a008014a408.html

[3] WSDL Tutorial : http://www.w3schools.com/wsdl/default.asp

[4] M. Rajagopal, E. Rodriguez, R. Weber : draft-ietf-ips-fcovertcpip-12.txt

[5] Ravi Natarajan, Anil Rijhsinghani : draft-ietf-ips-fcip-mib-05.txt
[6] Julian Satran, Kalman Meth, Costa Sapuntzakis, Mallikarjun Chadalapaka, Efri Zeidner : draft-ietf-ips-iscsi-20.txt

[7] Mark Bakke, Jim Muchow, Marjorie Krueger, Tom McSweeney : draft-ietf-ips-iscsi-mib-09.txt

[8] Dorothea Beringer, Harumi Kuno, Mike Lemon : WSCL et UDDI :
http://www.uddi.org/pubs/wscl_TN_forUDDI_5_16_011.pdf