User Controlled LightPath (UCLP)
Software review and equipment compatibility[1]

Remco Hobo & Ruben Valke

February 3 2005

[1]for the Advanced Internet Research group of the University of Amsterdam

Abstract

A User Controlled Lightpath is a bandwidth-on-demand solution; a way to get instant bandwidth over great geographical distances. All intermediate devices will be automatically configured for this connection. Three universities in Canada all made their own UCLP implementation, in this report we will discuss the usability of these UCLP implementations. Also we will answer the following questions:

- Is UCLP usefull in the Lighthouse lab?
- Is UCLP deployable in the lab with it's current equipment?
- What are the possbilities in a multi-domain environment?
- Can we also use UCLP with Layer1 network equipment like GlimmerGlass or Calient?

Contents

1 Introduction

UCLP is an initiative of Canarie[1]. Starting in 2002 they are searching for a solution to manage their optical network.

The UCLP software is designed to allow end users to create their own discipline or application specific IP network, particularly in support for high end grid applications. For example a community of high energy physicists researchers can create their own independent IP network (as a subset of a larger optical network) whose topology and architecture is optimized for their particular grid applications needs and requirements. More importantly these networks can be dynamically reconfigured at any time without getting permission or signaling the optical network manager.

The UCLP software allows end users to self provision and dynamically reconfigure optical (layer one) networks within a single domain or in the future across multiple independent management domains. Sometimes this is also referred to as user controlled traffic engineering. Users can also create daughter optical VPNs and hand off control and management of these VPNs to other users.

At this moment UCLP is not widely deployed and it can be considered to be in development state. Currently there are three different implementations of UCLP software, which one is going to be the standard version is not certain yet.

In this report we will give an overview of the development state of these projects and we will also answer the original research questions[1] asked.

[1]Canarie is the researchnetwork of Canada

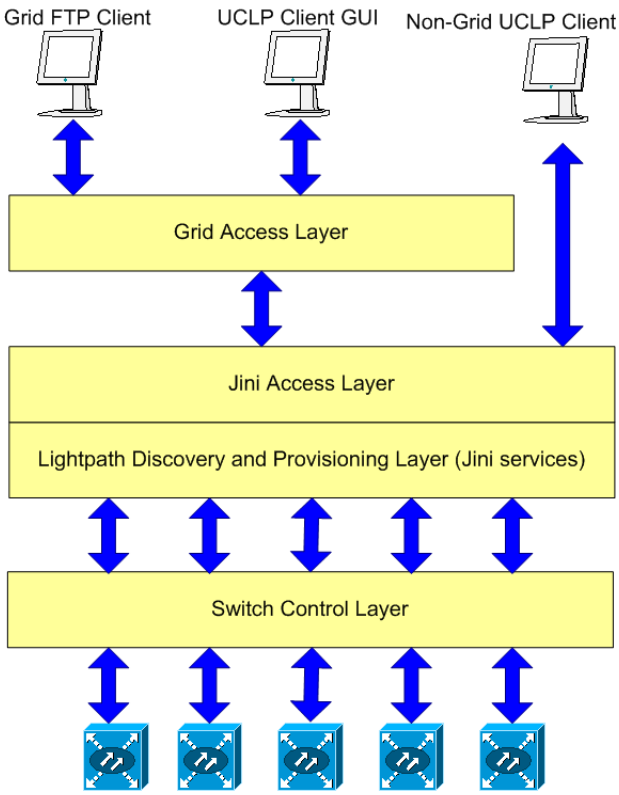
2 University of Ottawa

2.1 Theory

The software for the University of Ottawa consists of two parts:

- One for the GSAP (Linux).
  - Globus Toolkit 3.0;
  - Jakarta Tomcat 4.0.6 or 4.1.24 (tested with 4.0.6);
  - Jakarta Ant 1.5 or higher (tested with 1.5.1 and 1.5.3);
  - JUnit 3.8.1;
  - J2SDK 1.4.2.
- The other for the Jini services (Windows).
  - J2SDK 1.4.2;
  - The Jini package. (2.0)

The Linux part of this package is used to create a Grid Service Access Point (GSAP). This layer is used to connect to the package using Grid-enabled clients. The Grid-layer is not mandatory; there is also a possibility to connect directly to the Jini Access Layer (JAL) and bypassing the need to install a Linux server.
Figure: Ottawa overview



The Windows services can be implemented on one machine but can also be distributed across different machines. The two figures below describe how this can be set-up.

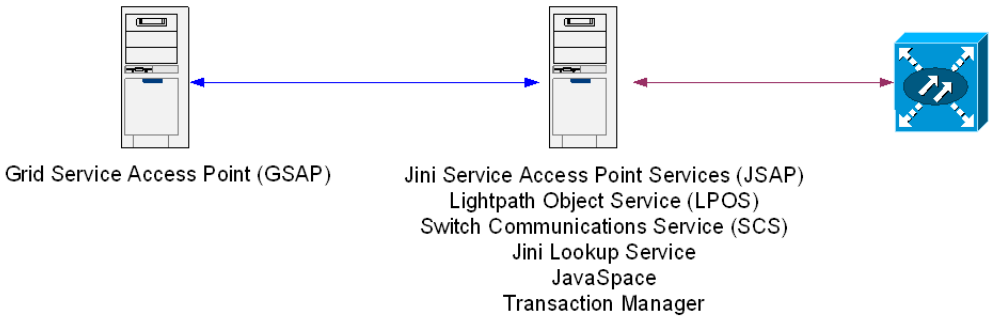


Figure: Ottawa windows servers on one machine

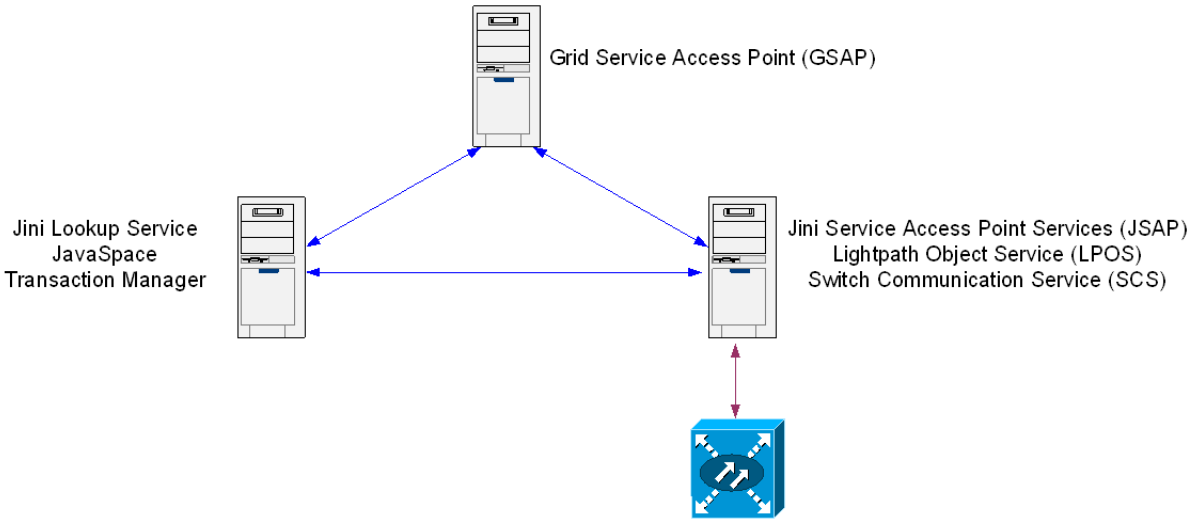


Figure: Ottawa windows servers on multiple machine

The Jini Lookup Service and Jini Service Access Point Services, like the name already gives away, make use of a Jini based services. Jini is a:

- Infrastructure and programming model which allow devices to connect with each other to create an instant community. Jini technology enables devices to work with each other, so users can create their own personal networks or communities no matter where they are located.;
- An open software architecture that enables the creation of network-centric solutions which are highly adaptive to change;

Glossary:
The Jini Lookup Service: The central component of Jini's runtime infrastructure offers Jini clients a way to find Jini services;

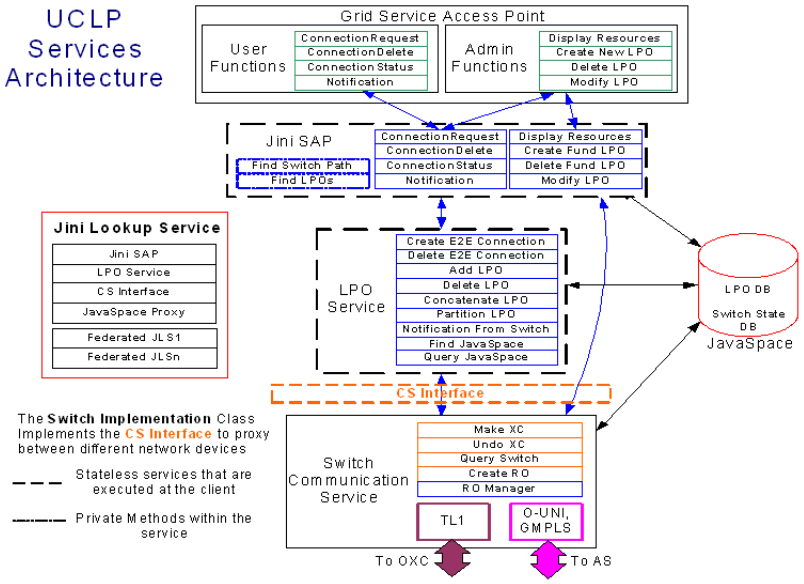
JSAP: Jini Service Access Point Services: A Jini service that acts as the gateway between the GSAP and UCLP core Jini services, such as LPOS and SCS;

LPOS: A Jini service that manages both fundamental LPOs and E2Econnection objects that are owned by the federation;

LPO: LightPath Object: The abstract representation of an optical link between two adjacent federations;

SCS: Switch Communication Service: A Jini service that concatenates/unlinks 2 fundamental LPOs, by making/undoing a cross-connection on an optical switch or setting up/tearing down an internal LightPath across an optical cloud, whichever applies.

The figure below will describe the complete architecture of Ottawa's university.



2.2 Practice

The installation of the Linux side is pretty straight forward but time consuming, especially the Globus toolkit 3 takes a long time to compile; as much as four hours are needed on a 500 MHz machine. Also, links included in the document for downloading pieces of software are dead; see the download section of this document for a working link.
Make sure the exact version of the J2SDK (1.4.2) is used, otherwise the GT3 will not compile properly.
After the installation has been completed, tomcat can be started. This web server is visited; the client software will automatically be downloaded. For this, a Java runtime has to be present on the client machine. When the client software starts up, a username and password can be filled in. Since the Windows side hasn't yet been set up, no user accounts exist.

The Windows side of the software is a whole other story. The complete installation manual consists of about two pages of instructions, which are nothing more then a best-case-walk-through. After installing all required components and running the UCLPConfigTool, which sets up all configuration files, the whole system should be able to start up using the wrun command included in the UCLP distribution, but it keeps complaining it cannot find a JLS (Jini lookup service). The

following services have to be started to get the system working:

1. httpd: The codebase server for the Sun Jini services.
2. phoenix: RMI Daemon - (Optional, only required if using activatable services.)
3. reggie: Jini Lookup Service with correct configuration to reflect the federation it will be a member of.
4. mahalo: Transaction Manager with correct configuration to reflect the federation it will be a member of.
5. outrigger: JavaSpace with correct configuration to reflect the federation it will be a member of.

Because the far from complete instruction manual, and because the Waterloo implementation looked more promising, further efforts to get this implementation to work were not taken.

3 University of Waterloo

This chapter describes the package delivered by the University of Waterloo.

3.1 Architecture and security mechanisms implemented

A detailed architecture of the system, consisting of three layers, is presented below.

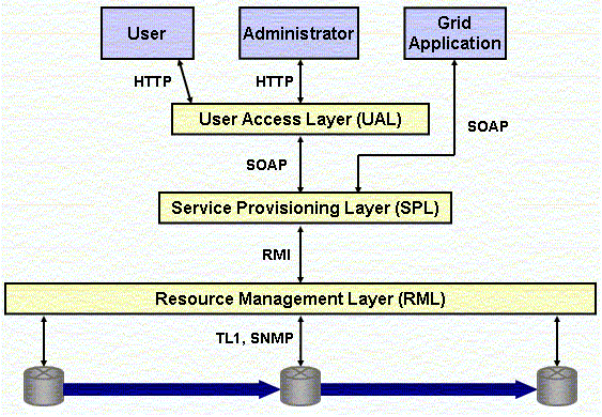


Figure: Waterloo overview

The user access layer (UAL) is concerned with handling and parsing requests from the Web-based interface used by customers and administrators.

The service provisioning layer (SPL) consists of a set of Grid services that implement functionality related to Lightpath manipulation. The security between UAL and SPL adopts the security mechanisms provided by Globus Toolkit 3.0. Mutual authentication is implemented using Transport Level Security and encryption is implemented using Message Level Security (XML encryption and XML signature). The CA authority and GSI security environment has been established and these features are activated. The machine running UAL is the client and the machine running is the host. They can be the same machine too.

Finally, the resource management layer (RSL) consists of a set of Resource Agents and a lightpath object (LPO) database. The Resource Agents perform low-level communication with the physical layer, i.e. network hardware and provide a virtualization of hardware resources, allowing customers to control the subset of resources dedicated to their own lightpaths. Lightpath objects represent lightpath-related data and are stored in the LPO database.

The figure below shows the permissions for various users:

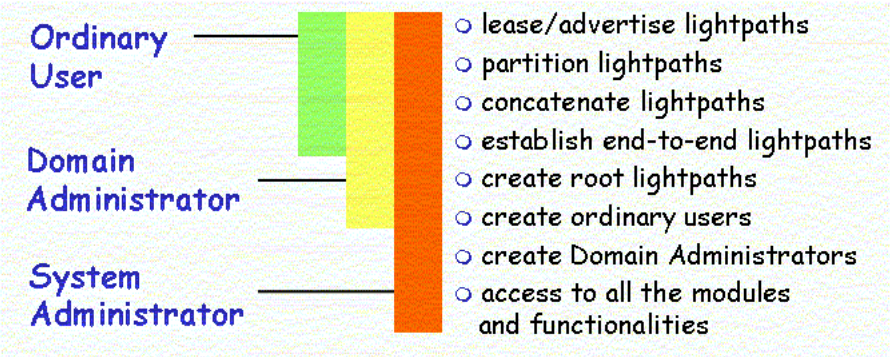


Figure: Ottawa user overview

As the figure above describes, a system Administrator can define multiple domains and then create domain administrators. These domain administrators can create root lightpaths, create ordinary users and do anything an ordinary user can:

- Advertise: This function is used to advertise an LPO, enabling other users to subsequently lease it. The status of the LPO must be RESERVED beforehand, and is changed to AVAILABLE. Only the current owner of an LPO can advertise it. An advertisement period is specified such that at the end of the period the calling user is guaranteed to regain possession of the LPO. A LPO must be partitioned before it can be advertised. In particular, this applies to root LPOs, and to LPOs that have been leased from another user. Also, the advertisement period must be shorter than the period for which the parent LPO has been leased. When the advertisement period expires, the status of the LPO is changed back to RESERVED. Refer to the related Reconfigure LPO function for information on how to apply policies concerning the usage of an LPO before advertising it for lease by others

- Lease: This function is used to lease an LPO, which involves transferring ownership from the user who advertised it to the user making the request. The status of the LPO must be AVAILABLE beforehand, and is changed to RESERVED. A lease period is specified such that at the end of the period the calling user automatically releases possession of the LPO. In that case, the status changes back to AVAILABLE and ownership is transferred back. The lease period must be shorter than the period for which the LPO has been advertised. Once an LPO is leased, it cannot be advertised again. Instead, it must be partitioned, and the children can be advertised. The user may select not only the end of the lease period, but also its beginning, which allows a user to request an LPO in advance.

- Access: This function is used to cross-connect the endpoints of a lightpath to Ethernet ports. The user must select the Ethernet card and port to be accessed at either end from a drop-down list. An LPO can only be accessed if its status is RESERVED. Its status then changes to ACCESSED.

- Partition: The partition operation can be performed on an LPO whose status is RESERVED. The user selects the parent LPO, as well as the bandwidth of the desired child lightpath from a drop down list. Subsequently, the system reports additional child lightpaths that must be created in order to satisfy

bandwidth granularity constraints. Finally, the user confirms and the full set of children is created, each represented by a new LPO. The status of the parent LPO is changed to INACTIVE, while the child LPOs are initialized with a status of RESERVED. The lease expiry date of the parent LPO is inherited by the child LPOs during the partition operation.

- Concatenate: To concatenate a set of constituent lightpaths, the user selects the constituents through a search by bandwidth. This guarantees that all the constituents have the same bandwidth. Furthermore, the search results are filtered so that only LPOs with a status of RESERVED are eligible for concatenation. Next, the user sets the order of the constituents and submits the request. The system then ensures that all the constituents have the same directionality, and that they can in fact be concatenated (e.g. every consecutive pair of constituents in the list must share an endpoint). Finally, the status of the constituents is changed to INACTIVE, and a new compound LPO is created with a status of RESERVED.

- Create End-to-End lightpaths: To establish an end-to-end LPO, the user specifies the following information:

  - source and destination cross-connect devices
  - the routing engine
  - any options specific to the routing engine, for example the routing metric when using shortest path routing, Dijkstra's Shortest Path is the only routing engine available at the moment.
  - the required bandwidth
  - whether the routing computation should ignore lightpaths that have more bandwidth than the user's requirement (i.e. 'use exact bandwidth')
  - the lease period, in case some of the constituents must be leased
  - the directionality of the desired lightpath
  - whether LPOs advertised by other users should be considered by the routing computation in addition to the user's own lightpaths having a status of RESERVED or AVAILABLE (i.e. 'Use advertised LPOs')

  After submitting the request, the user is offered a candidate end-to-end path, consisting of a list of constituent LPOs. If the user accepts the candidate path, the system constructs the end-to-end LPO as follows:

  - any constituents that do no not belong to the calling user are leased
  - any constituents that belong to the calling user but have a status of AVAILABLE are reserved by cancelling the advertisement
  - any constituents that have more bandwidth than the user's requirement are partitioned
  - any excess children created in the last step are advertised for a period of time equal to the specified lease period
  - the constituents or partitions thereof are finally concatenated

In the event that the end-to-end path consists of a single constituent, the concatenation step is skipped.

Glossary:

LPO: Lightpath Object
OGSI: Open Grid Services Infrastructure
RML: Resource Manager Layer
SONET: Synchronous Optical Network
SPL: Service Provisioning Layer
UAL: User Access Layer
UCLP: User Controlled Lightpaths Project
WDM: Wavelength Division Multiplexing

3.2 Practice

The hardest thing is acquiring all the software and documents. For this, multiple license agreements etc. have to be filled in; afterwards credentials will be mailed to you. On the second try, we got a response with the requested credentials. With these credentials, software and documents can be downloaded. These documents include an installation manual consisting of 25 pages of instructions for setting up al the different components.

In this manual, thankfully, links are given to the exact versions of all components so incompatibility issues will be greatly reduced. After untarring the Waterloo package to a Linux server, the different packages like JBOSS, Globus Toolkit, COG etc. have to be downloaded and untarred to the archives folder. On a Redhat server, the 'installMySQLAsROOT' script can be executed. On a Debian machine, please use alien and then dpkg to install the MySQL server and client. Use the exact same version as listed in the document otherwise the package might complain it cannot connect to the database. After this, the script 'install' can be run. WARNING: DO NOT RUN THIS SCRIPT ON A MACHINE THAT HAS OTHER DUTIES AS IT CAN BE VERY DESTRUCTIVE TO PRECONFIGURED THINGS LIKE APACHE, MYSQL, JAVA ETC.

When all components are installed, they have to be configured. A simpleCA as a trusted authority has to be created. For this, a script can be downloaded from the Waterloo UCLP website which will do this for you. After executing some complimentary commands, the CA is installed and certificates are made for the different components. If the UAL and SPL are on different machines, a GT3Proxy can be used in combination with a secure GT3 Container. If both are on the same machine and the ports are blocked for non-local traffic, an insecure GT3 Container can be used.

To get the secure container to work, execute "cd $GLOBUS_LOCATION; ant setup;", this step is overlooked in the manual.

Setting up the agents turned out to be the hardest. The agents aren't very talkative; they will just exit when something isn't setup right. Also, if you fill in an IP address in the agent.properties file and add an extra space at the end of the line, the agent crashes on this. Also, when the certificates aren't installed properly for the agent, this will result in an exit without a reason. Please read the part about generating certificates CAREFULLY and act according to the instructions.

The agents are used to connect to a NE (Network Element). Every agent will connect to one network element. Each of these agents can be configured for each

specific NE through an xml file, specifying the installed blades, interfaces, bandwidth etc. It is also possible to configure the agents mimic a Cisco ONS15454 with a G1000 card in slot 4, an OC-12 card in slot 5, and an OC-192 card in slot 13.

The installation of Waterloo's implementation is time consuming and you have to pay attention to the details. If you follow the installation manual to the letter, you should be able to install it without much difficult. If you run into problems, the guys at Waterloo's university are quite happy to help you.

Some installation notes:

- The path to gunzip (/usr/bin/gunzip) is hardcoded in the installation, for Debian; create a symlink from /bin/gunzip to avoid getting into trouble.
- Keep the ports rmi.port and http.port in the agent.properties file unique if you plan to run multiple instances of the agent.

3.3 Look and feel

The GUI looks quite well-designed with nicely ordered menu on the left hand side. However, it very quickly becomes clear that it is still work in progress. The top navigation bar isn't yet active. After getting used to the menu, it works quite well. Getting an overview of current leases etc. could be improved. To get an overview, now you have to go to cancel lease, and then search for an empty string as source. Also, it would be nice if all available options for bandwidth would be implemented in a drop-down box. At the moment you have to fill in a number like 51840 for STS-1 and 466560 for STS-9. Also, you have to fill in the channel selection yourself, and this can be complicated for new users. For instance, if you use STS-1, you can select any channel: 1,2,3 etc. For STS-9 you can only select 1,10,19,28 etc. Also, there is no easy way to see which channels are occupied at which NE, so it can be a hassle to set up a RO.
Also, when a error occurs (bandwidth not supported, channel number incorrect, channel occupied etc.) this will only be shown on the console from where the services are started. The GUI will only indicate the action was unsuccessful.

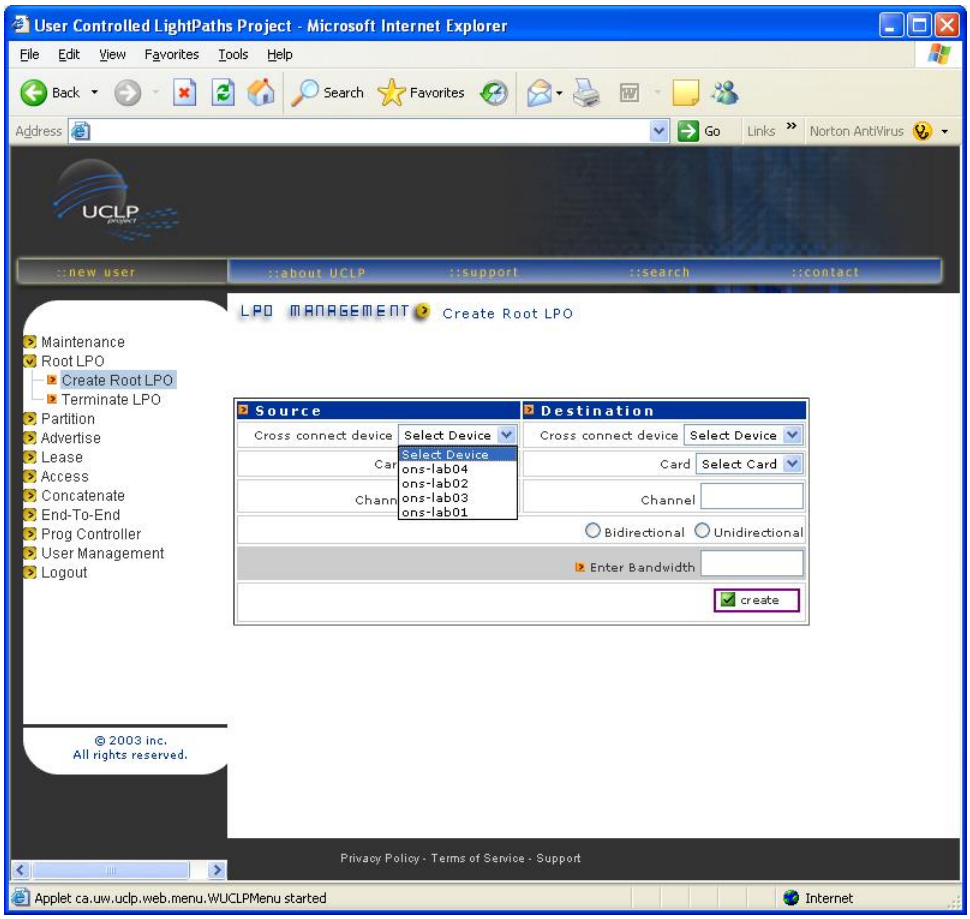
Challenges for next version:

- Make the GUI more user-friendly.
- Get the search option active.
- Improve feedback to the GUI

Figure: Waterloo GUI

3.4 Security

Users will connect to the GUI which is hosted on a Tomcat webserver, this way, unsecured http is used. However, another option is to incorporate the GUI into apache, thus enabling the https protocol.
Communication between UAL and SPL is done using security mechanisms provided by Globus Toolkit 3.0. Mutual authentication is implemented using Transport Level Security and encryption is implemented using Message Level Security (XML encryption and XML signature).



Communication between the SPL and the agents is encrypted using a RSA 1024-bit certificate, which is stored in the java keystore. When JBOSS is restarted, the certificate will be imported into it.
Communication between the agents and a NE can be done plain text, which is fine as long as the connection is shielded form the outside. If not, SSL can be used to communicate with the TL1 agent.

3.5 Tests

After some initial trouble with the agents and credentials, we where able to communicate with the two Cisco ONS15454's over at Canarie's test lab in Canada. We connected to them directly with and without the use of the TL1 agents. These agents are a kind of proxy in front of the NE's, a sort of last line of defense against flaws in one of the implementations. They will block requests that can be dangerous for the running configuration. In both methods we were able to login and issue commands like creating LPO's. The downside was that the Cisco's where naturally preconfigured by other people using it as a test bed. Due to this, we got some errors indicating that ports where already configured:

> The UCLP System has detected a problem with some cross-connections on the switch associated with resource agent ons-lab01. some cross-connections are present on the resource agent ons-lab01. The card

G1000-4 in slot 4 has a cross connection on the following port(s):[2]. The ACCESS Operation will not work on these ports unless you delete these cross-connections by hand.

Since we'd prefer to work on a clean environment, we already where successful in communicating with the lab at Canarie and the dummy Cisco's work the same, we decided to use these instead.

For testing purposes, we have set up four dummy agents, Cisco ONS 15545's with a G1000 card in slot 4, an OC-12 card in slot 5, and an OC-192 card in slot 13. This is the only dummy NE available.

- Creating domains: This is easily done, but when you add the same domain twice, there will not be an error message; only on the console one will be present, not in the GUI.
- Creating domain administrators and users: Works fine.
- Create LPO's: Adding LPO's works. It however would be nice to know what options can be set and which ports are occupied etc. When a LPO is added, this will be added to the MySQL database but also to the agent. An object will be added in a subdirectory of the agent.
- Partition the LPO's: Before a LPO can be advertised for use, it needs to be partitioned. If you want to advertise the full bandwidth of the connection, you can partition it into one child LPO. Otherwise you can partition it in STS chunks. All partition attempts worked fine.
- Lease the LPO's: If an administrator advertises a connection so it can be leased, you would like to know until when it is. This information is not available. Also, a lease can be scheduled so the start and end time can be in future. But when you do this, there is no way to cancel the lease until it becomes active, not even for the system admin. This way, the LPO becomes 'frozen' until its start time is reached. This is a major bug. Misuse this once and the LPO and even its root LPO can not be changed or removed until the start time is reached.
- Access LPO's: Accessing a LPO can be done after a lightpath has been leased. This way you can configure the data connections on the NE's. This is the connection to the GigE ports, and final delivery of data. This works fine but it isn't schedulable. It would be nice if you can schedule a lease, and then schedule a access for it, to the whole topology change could be automated.
- Concatenate LPO's: This way, lightpaths can be glued together to form one longer lightpath. It works as expected.
- ETE lightpath: An End To End lightpath makes it possible to just give in a source and destination NE and let the application figure out how to get there. At the moment it only knows Dijkstra's shortest path algorithm. You can also schedule an ETE, so the lightpath will be available at the required moment in time. A user also has to fill in bandwidth requirements. When 'use exact bandwidth' is unselected, the system will also use LPO's with bigger bandwidth then demanded. It works as expected but again, when a LPO is scheduled, the LPO can not be changed, removed, etc.

Challenges for next version:

- Fix the scheduling for LPO's. Make sure users can schedule use of a lightpath for multiple time boxes.
- Make cancellation possible for lightpaths that are scheduled
- Create scheduling options for accessing LPO‘s.

3.6 Conclusion

The Waterloo implementation of UCLP has great potential, it is still in it‘s early development but all main functionality is operational. The package is reasonably easy to install, the only thing that needs attention is the agent. It should at least give a reason why it will not start instead of just stopping. Also, the user friendlyness of the GUI can greatly improved, but in this early development stadium it already shows great potential. Since it still is in early development, it can only handle ethernet and SONET, it has no knowledge yet of any other structures like 802.1p/q as mentioned in the concept UCLP document. When the impementation is used in conjunction with Apache, the security is guarantied. The implementation is quite stable, accessing a lightpath is the only thing that sometimes triggered an internal error, everything else always worked as supposed to.

4 Universitè du Quibec à Montreal

4.1 Introduction

The OPTICNET group of the University of Quebec and Montreal[2] also proposed a project proposal to the UCLP project of Canarie. Canarie selected the UQAM project together with three other projects.

From the four projects, the UQAM project is the only one witch implemented the OBGP-standard into its project. The OBGP implementation is the most eye-catching part of the UQAM release. OBGP makes it possible to calculate the shortest path between two points further on in this chapter we will discuss the OBGP possibilities in more detail.

4.2 software architecture

All the UCLP releases used a different software architecture, to understand all the different terms we will start to explain what alle the software element do and what there functions are. In the next diagram, you can see the correlation between the different elements.

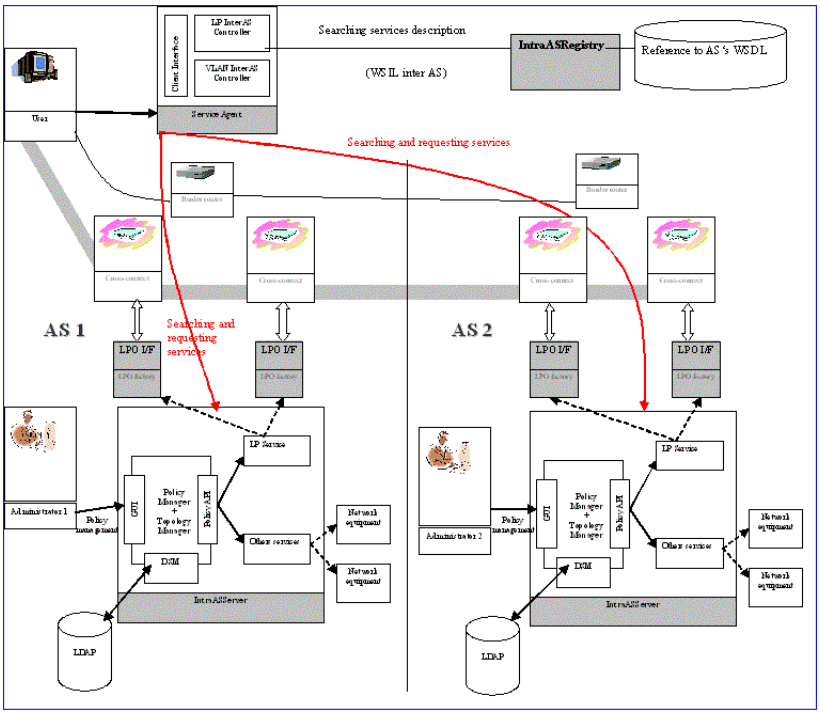


Figure 1: UQAM architecture overview

In the above figure the system design of the whole UCLP system is shown. three distinct main control elements can be found, these three elements are:

- ServiceAgent
- IntraASServer
- IntraASRegistry

[2]from now on referred to as UQAM

In the next paragraphs these elements and its components are discussed in more detail.

4.2.1 ServiceAgent

The service agent is the access point for the UCLP system it functions as a layer between the InterASServer and the InterASRegistry for the end user. The end user GUI communicates only with this serviceAgent. For instance if a user requests an inter-domain lightpath the serverAgent will lookup the BGP route to the end-point. If an appropriate route is found, the serviceAgent will ask all the involved intraASServers to provision the lightpath. Also the ServiceAgent will notify the user if anything changes in the lightpath. Next to the end user GUI, this ServiceAgent communicates with the IntraASSERVER and IntraASRegistry via a WSDL interface.

4.2.2 IntraASServer

Within the boundaries of one domain, the IntraASServer manages all lightPath settings. The services within the IntraASServer are configurable via the consoleAdministrator tool and communicate with each other via WSDL. An overview of the offered services:

- policy manager
- topology manager
- lightPath Services

Policy manager: The policy manager managers the domain policies for its domain, this means for instance that the policy manager checks if all AAA goals are met. Also, the policy manager keeps an inventory of all equipment within its domain.

Topology manager: The topology manager helps the policy manager and the LightPath services. If a user requests a search for a lightpath, the topology manager will search the route between two interfaces. If the topology manager finds a matching route, that route will be passed on to the LightPath service. The LightPath service then can deploy the LightPath.

LightPath services: LightPath services are provided by the LPServer. The LPServer can create and delete LightPath Objects on a CrossConnect device. The LPServer stores all defined LighPaths in the available LDAP server. This way a LightPath cant be deployed twice

4.2.3 IntraASRegistry

The IntraASRegistry is a repository for AS numbers, it links an AS number to an InterASServer webservice. witch controls one or more. There has to be only one IntraASRegistry for all lambda networks. The IntraASRegistry provides his services via a WSDL interface.

4.3 User interfaces

In the previous section, we discussed the different services offered. These services do not have a user interface, so an end user does not have to know they are there. We can divide the offered user interfaces in three administrative groups:

- End user; a user who requests lightpaths.
- Administrator; a user who administers a particular domain (one AS).
- AS administrator; an administrator who maintains the AS repository.

	End user	Administrator	AS Administrator
UCLPGUI	X		
Console Admin		X	
- Policy Manager		X	
- Blocking STS chanels		X	
Topology Manager		X	
IntraASregistry			X

4.3.1 UCLPGUI

The UCLPGUI shown in figure 2 is the only end user application. Via this tool an end user can create, add, edit and delete his (intra domain) lightpath. The interface is userfriendly, and ratively simple. Reletively since since the user still has to understand some SOnet abbreviations, and the principles behind SOnet and Ethernet over fiber, but more in this in de evaluation chapter.

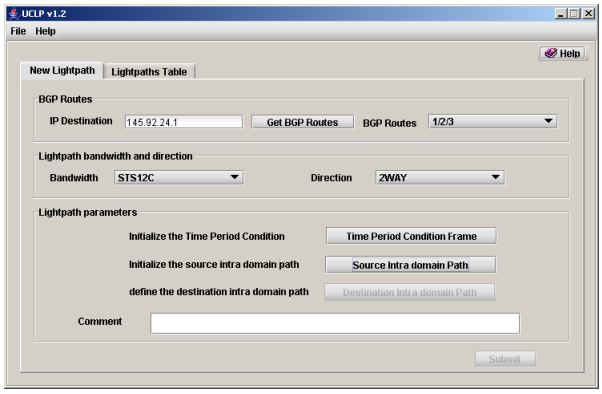


Figure 2: UQAM uclp gui

4.3.2 Console Admin

The main part of the consoleAdmin (figure 3) is to configure all components, so this has to be accessed only once or twice by the system administrator. But two elements will probably be used on a more regular basis: The PolicyManager and the Blocking STS chanels function.

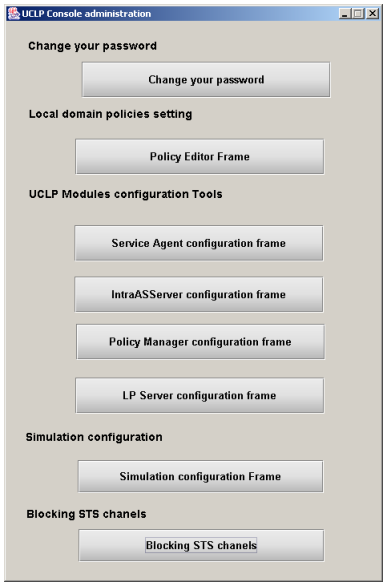


Figure 3: UQAM ConsoleAdmin

Policy Manager: The policy manager (figure 4) manages the domain policies for its domain, this means for instance that the policy manager checks if all AAA goals are met. Also, the policy manager keeps an inventory of all equipment within its own domain.

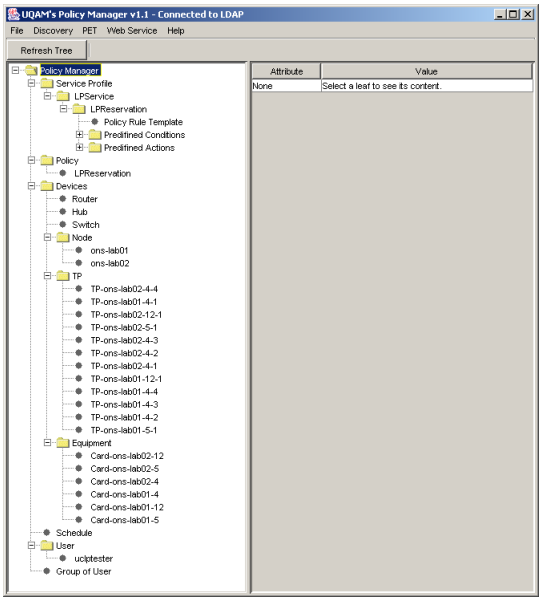


Figure 4: UQAM Policy Manager

Blocking STS chanels: The STS blocking function(figure 5) lets an administrator block a specific part of an SONET link. This function is for example usefull to keep a lightpath from being leased. This way the same equipment can be used to handle both the requested LightPaths and static production Lightpaths (e.g. their normal internet connection).

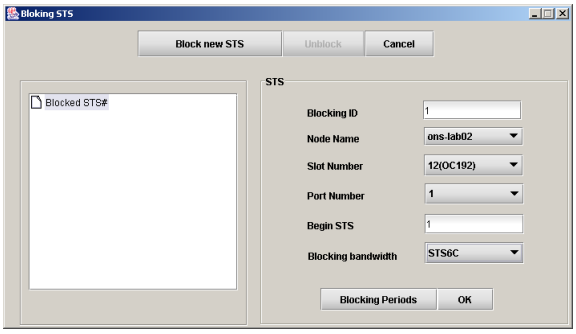


Figure 5: UQAM blocking STS chanels

4.3.3 Topology Manager

With the topology manager (figure 6) a manager can specify inter domain connections, this means a connection that leaves the bondries of his domain (e.g. AS1) and enters the the next domain. For instance, you can specify an OC-192 connection from an optical cross-connect device in Amsterdam (e.g. the owner AS1) to an OC-192 connection of an optical cross-connect in Los Angels (e.g. AS2).

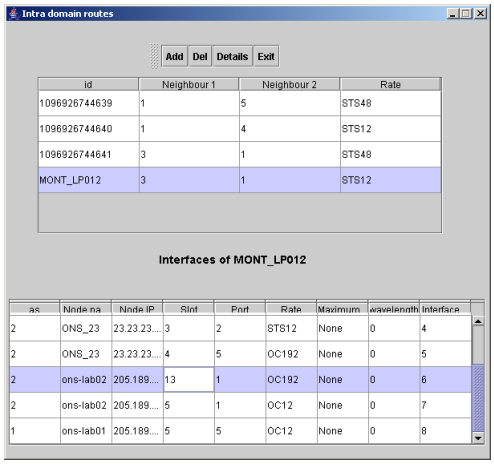


Figure 6: UQAM topology Manager

4.3.4 IntraASRegistry

The IntraASRegistry GUI shown in Figure 7 is a small GUI, via this GUI the AS repository can be updated. The URL you see is the pointer to the webservice controlling a IntraASServer. Since there should be only one IntraASRegistry global, only one entity should control this repository.

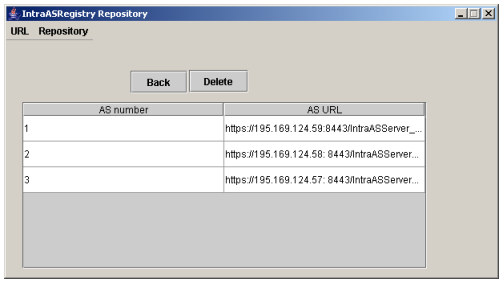


Figure 7: UQAM IntraASRegistry

4.4 Conclusion

In this section we will give our opinion about the UQAM UCLP implementation. Since there are a lot of points to discuss, we divided this chapter into separate paragraphs.

4.4.1 User interfaces

The different user interfaces work fine and all have a good look and feel. But we think that it would have been better if the end-user interface (UCLPGUI 2) was implemented as webbased application in stead of a java windows application.

The users who will use the UCLPGUI, are researchers around the globe (although a user will only connect to a service agent within his administrative domain). A webbased UCLP is mutch easer to access, and users are used to such interfaces. The UCLPGUI depends on an old Java JDK, and the research users needs enough rights on his machine to install the GUI. Also de UCLPGUI is an .EXE, so it can only be used under the Windows Operating System.

The other interfaces (e.g. the console admin) are also a java applications, since an domain admin will use this interface only on one or two machines, it is less of an problem that it is an java application.

In general I think it is better if they make the user interfaces more OS independent. The majority of the end users will be researchers. Researchers use a lot of different Operating Systems, and all the windows dependant software will hold the researchers back for using UCLP.

4.4.2 Installation

The software package is accompanied by an installation manual. This manual is sufficient if you know how the system is build up, and if you anticipated in the development. This means that the manual was too way to minimal.

Before we had the version working, we had to gain knowledge of java, jsp, tomcat, wsdl, ldap an SOnet. After a while almost all parts where working, except for the end user interface (UCLPGUI) this interface just wouldnt connect to the LDAP database. After invaluable help of Ghandour Boubker[3]we got the system to work. The problem was that the java JDKwe used was too new. We used JDK1.4.2_06 (recommended by the installation document) but the UCLPGUI would only work with JDK1.4.1_06. During the installation, we noticed some more inaccuracies in the installation document, these small errors (witch took a lot of time)can be found in appendix B the installation manual for this software release.

4.4.3 Look and feel

One you have managed to install UQAM UCLP, it is simple to use. Of course you should have basic knowledge of networks, but I think that anyone who will work with UCLP knows this, except for the SOnet part. I dont think a lot of people will understand all abbreviations like OC-192 and STS when they start to work with UCLP as an end customer. Knowledge of SOnet is mandatory since almost all long distance high speed connections are based on this technique

4.4.4 Implementation

This software implementation is the closest to the original Canarie design of all three implementations. For instance the UQAM UCLP implementation uses OBGP[4] to exchange intra-domain routes between domains. Also the development is going very fast, within this year there where already two releases and one of the developers told me that a new version will be released very soon.

If the development team can manage to make the software less OS dependent, this system could be make it as the de-facto UCLP standard.

[3]Ghandour Boubker is one of the developers of this UQAM UCLP version.
[4]OBGP is currently still a draft version

5 Recommendations

Within this chapter we will answer and discuss the research questions. These research questions where stated during the first week of this project.

5.1 Is UCLP deployable in the lab with its current equipment?

The current implementations of UCLP only use SONET type equipment. To get the GlimmerGlass[5] working with the Waterloo implementation requires a rather dirty workaround. All implementations have only yet developed software for SONET, and not yet things like Gigabit Ethernet VPN with dedicated bandwidth (e.g. using 802.1p/q[6]) which is mentioned in the concept of UCLP. Waterloo only accepts SONET STS connections from and to the same channel of an OC card. This means you can not define each blade of the Glimmer Glass as a different card and define all the different connectors as channels. You have to define all different interfaces as different cards and then fill in the same channel number as the source OC interface when setting up a Light Path.

To control the glimmerGlass via UCLP, a TL1 proxy has to be developed, that will translate the TL1 commands from the Waterloo implementation into correct TL1 for the Glimmer Glass, and thus ignoring the bogus channel numbers.

The Force10 core router doesnt speak TL1 itself but it does know SNMP. However, the Force10 doesnt use SONET but uses 802.1p/q to make a path from one interface to another, which is incompatible with SONET. An agent could be developed for translating the TL1 commands to SNMP quite easily but the translation from SONET to 802.1p/q will be quite difficult and is also dirty. For instance 802.1p/q can assign any bandwidth to a Vlan but SONET STS has predefined bandwidths. Also, with SONET you have to fill in channel numbers, which have a granularity corresponding to their bandwidth. We think its better to wait until 802.1p/q is implemented into one of the UCLP implementations before trying to implement it into the Lighthouse.

[5] GlimmerGlass is an optical crossConnect , witch connects fibers to eachother. the signal is processed via mirrors rather than via an optical-electrical-optical translation

[6] .1q = vlan tagging, .1p is traffic Prioritizing (Quality of Service)

5.2 Is UCLP useful in the LightHouse lab?

The main reason for using UCLP in the Lighthouse is to be able to multiplex the 1Gbit/10Gbit interfaces, since these can be quite costly. For this, the Glimmer Glass can easily be used. The main reason for using UCLP in general is to be able to create easily manageable paths over a wide geographically area. The current Lighthouse has a limited number of users and topology changes are not carried out on a daily basis. Therefore a much easier solution might be used, like writing a simple script for scheduling topology changes or using the Glimmer glasss own GUI. Due to the incompatibilities mentioned above and the current use of the Lighthouse it is our opinion UCLP is not useful at this moment but might become useful in the future, when the Lighthouse might be used by more users and in a larger scale.

One reason to start using UCLP in the lighthouse could be the arrival of the cisco 15454 boxes in the lighthouse lab. All UCLP releases are based on this equipment and also the UCLP design is based on this type of equipment.

If a glimmerglass agent is developed, the glimmerGlass can be configured to have SOnet compatible lightPaths to multiple external endpoints. Internally a lightPath can be created to a ONS switch. By concatenating an external lightPath and the internal lightPath, a connection can be established through the glimmerGlass. When a lightPath has to be created to another external endpoint, the concatenated lightPath can be partitioned again and the new external lightPath and the internal lightPath can be concatenated again.

5.3 Multi domain posibilities

One reason to start using UCLP in the lighthouse could be the arrival of the cisco 15454 boxes in the lighthouse lab. All UCLP releases are based on this equipment and also the UCLP design is based on this type of equipment.

In the original design documents of UCLP, there was already a chapter about inter domain LightPaths. Later on in the UCLP development phase, the *Optical Border Gateway Protocol* was designed. OBGP makes is possible to exchange routing information, in case of an optical only network, these routes are lightPaths.

The power of the whole UCLP concept is for a great part the possibility to make lightPaths between different administrative domains. The strange think is that only one of the three releases we have tested is fully capable of making intra domain lightPaths.

The UQAM version has a fully working OBGP implementation, witch works pretty simple. In principle there has to be only one intraASRegistry. This registry stores two things, an AS number and a URL. This URL will point to the webservices of a specific domain. Intelligence at the serviceAgent within the users domain has to search the best route via this intraASRegistry.

The above solution is simple and it is also very effective. But is starts to get inefficient when a lot of AS numbers are registered at the intraASRegistry because the serviceAgent has to poll all the ASs webservices to investigate his available routes, on a large implementation, this could take a while.

The UQAM version is ready to implement in a multi domain environment, if the developers think of a other way to exchange the AS-path information. For the UCLP project it is best if the UCLP developers negotiate on a standard for exchanging this data.

References

[1] Start document of this project http://www.os3.nl/ ruben/rp1/rp.pdf

[2] OBGP design document http://www.canarie.ca/canet4/uclp/presentations/obgp_ietf.doc

[3] Java JDK Download location http://www.java.sun.com

[4] University of Waterloo http://bbcr.uwaterloo.ca/ canarie/download.htm

[5] Ottawa University - Communications Research Centre http://www.site.uottawa.ca:1090/uclp/

[6] Download link for University of Ottawa http://phi.badlab.crc.ca/uclp/downloads/downloads.html

[7] Universite Quebec a Montreal (UQAM) http://www.teleinfo.uqam.ca/opticnet

[8] Download link for University of Ottawa http://phi.badlab.crc.ca/uclp/downloads/downloads.html

[9] General UCLP information and presentations page http://www.canarie.ca/canet4/uclp/

A Installation procedure UQAM UCLP

A.1 Introduction

The University of Quibec a Montreal (UQAM) was one of the universities that made a UCLP implementation for the canarie UCLP project. This version is still in development, so not all features are available yet. The installation proces is pretty streat-forward, but laks details on some points. In this document we will discuss the missing details. Unfortunalty we didn't manage to get this version working, but we hope this document can give some help.

A.2 Pre installation requirements

- Java JDK 1.4.2_06 + java virtual machine
- Java JDK 1.4.1_06
- if not available a LDAP server (we used netscape iPlanet 5.0)

A.3 Java JDK installations

After lots and lots of research and help of the developers, we found out that the UCLPGUI would not run because we used the wrong JDK version. All applications within the UQAM UCLP are build with JDK1.4.2, except for the UCLPGUI. The UCLPGUI uses JDK1.4.1. Unfortunately This is not noted manual anywhere.

The installation of the two JDK packages is simple, just download and install them.

A.4 iPlanet 5.0 LDAP Installation

The installation of the LDAP server is not described in the manual, to ease the installation proces we will describe the settings we used.

- Select Server or Console Installation: iPlanet Servers
- Type of Installation: Typical
- Location of Installation: c:/iPlanet
- Components to Install: All (default)
- Configuration Directory Server: This instance will be the configuration directory server
- Directory to store data: Store data in this directory server
- General settings: Server Indentifier: UCLP-BOX Server Port: 389 Suffix: dc=uclp, dc=uva, dc=lighthouse, dc=nl
- Configuration Directory Server Administrator Configuration Directory Administrator ID: admin Password: Add your password
- Administration Domain Settings: admin
- Directory Manager Settings: Directory Manager DN: cn=Directory Manager Password: Add your password
- Administrator Port: 10000

A.5 Other small modifications

There are two main reasons why the UQAM UCLP release does not work out of the box:

- java JDK path pointer
- URL pointers to the webservices.

A.5.1 javaJDKpath

The following applications all have their own JDK pointer. Note that sometimes an end slash is necessary.

Application	JDK path
IntraASServer	c:/j2sdk1.4.2_06/
LP server	c:/j2sdk1.4.2_06
JAVA_HOME	c:/j2sdk1.4.2_06/
CATALINA_HOME	c:/progra 1/UCLP/jakarta-tomcat-1.4.24-LE-jdk14

A.5.2 URL pointers

Within the applications, there are pointers to the webservices. These webservices are provided by a tomcat application server via HTTPS. In our situation, all webservices where on the same machine, so we used the pointer: https://localhost:8443/webservice. But not all applications could cope with the localhost abbrivation. Later on we changed the pointer to: https://145.169.124.59/webservice. This is not a big problem, but it requires that the machine has a fixed ip-address.

B Installation procedure Waterloo

The installation manual is not public domain and can only be downloaded with the right credentials and with Permissions of the developers.