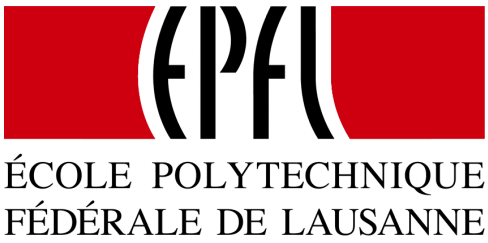
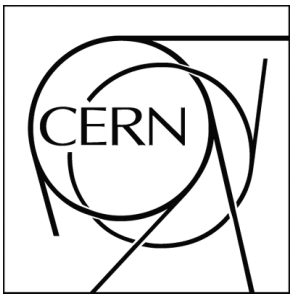




Reservation Application for a Transatlantic Gigabit Testbed

Diploma Project

Final Report (M.S. Thesis)

M.S. Student:

Simon Leo

Supervisors:

Prof. A. Schiper, EPFL

Dr. J.P. Martin-Flatin, CERN

Contents

1 EXECUTIVE SUMMARY 3

2 INTRODUCTION 4

2.1 CERN ORGANIZATION 4
2.2 GRID 4
2.3 THE DATATAG PROJECT 4
2.4 OUTLINE 7

3 TRANSATLANTIC TESTBED AND ITS CHALLENGES 8

3.1 SITUATION DESCRIPTION 8
3.2 INTERFERENCES USING THE TRANSATLANTIC TESTBED 9
3.3 THE ARCHITECTURE OF THE RESERVATION APPLICATION 10
3.4 DESIGN CHOICES 11
3.5 THE RESERVATION APPLICATION IN MORE DETAIL 13
3.6 TECHNICAL CONCEPTS OF THE CLIENT 15
3.7 TECHNICAL CONCEPTS OF THE SERVER 19

4 CONCLUSION 27

4.1 SUMMARY 27
4.2 WHAT DID I LEARN 27
4.3 FUTURE WORK 28

5 ACKNOWLEDGEMENTS 29

6 APPENDIX 30

7 REFERENCES 31

1 Executive Summary

This report describes a project where a management and reservation application has been developed in the context of a Transatlantic Gigabit testbed whose endpoints were located in Chicago and Geneva. The testbed was used by research groups who tried to make experiments using the link simulating traffic produced in the context of High Energy Physics where huge amounts of traffic were generated using high rates. Therefore researchers produced traffic to perform measurements. With the data received at the other end of the link they tested their algorithms. To generate the traffic Linux PCs where connected at both ends through a router at each side. Since the researchers did not have local access to the link but access from distance using the Internet, interferences between different users of the link occurred. Some of the users changed configurations of the Linux PCs while another person, who had launched an experiment, used them. Others rebooted the machines to install their own kernel without checking if someone was also using the same PC. Because of such interferences happening in uncontrolled manner researchers had to conclude that the results obtained were meaningless.

Users were geographically dispersed. Because of that and the occurring problems an application to handle such situations was designed and implemented. The application allowed users to reserve the link using a centralized Web interface. This application also managed access to the PCs generating and receiving traffic going through the link blocking or granting access to the PCs while some one had reserved the link.

To develop the reservation application free tools were used in a 3-tier architecture. Using a web server, a servlet container, Java Applets and Unix Shell Scripting a reservation application has been developed. Before there was any reservation application, chaos ruled access to the link. People tried to inform each other about the use of the link by email. But this turned out to be not very efficient. Thus the following project has been brought into life.

2 Introduction

2.1 CERN Organization

The whole project was done at CERN (European Center for Nuclear Research). CERN is a joint venture between 20 member states in Europe. It provides a testbed in the domain of experimental physics. Its main purpose is to find out what the matter is made of and what holds it together. Therefore CERN provides a particle accelerator for researchers in experimental physics. At the moment the particle accelerator is not running. CERN is constructing and installing a new particle accelerator, the Large Hadron Collider (LHC). This new collider, once it is finished, will produce large amount of data (Petabytes). This amount of data is so big that no local site at present can neither store the date nor process it.

2.2 Grid

To solve this problem the next generation of the Internet was about to be created. It was called Grid. Similar to peer-to-peer applications like Napster or Kazaa, which share storage capacity over the Internet, Grids were also designed to share storage capacity. But not only storage should have been shared using Grid applications, also processing capacity. Both resources were planned to be shared over the Internet in a transparent manner such that it was not obvious where particular data was processed or stored.

2.3 The DataTAG Project

For the Large Hadron Collider at CERN a new project was created to do research in the Grid field. It was called DataTAG. It is funded by the European Union. The work within DataTAG is dedicated to three different subjects.

The first one is to provide a transatlantic testbed to allow researchers in the Grid domain to test their algorithms on real hardware. DataTAG provides a testbed for high-speed networking. The latest hardware has been installed in the United States as well as in Europe. This is the testbed where the work described in this report belongs.

The second goal of DataTAG was to increase performance of existing network protocols. The actual protocols do not scale very well using gigabit network hardware. One of the existing protocols that work with bad performance compared to the theoretical maximum is TCP. Its “slow-start and congestion avoidance” algorithm does not work in the high bandwidth networks. After every packet loss its “slow start”-algorithm tries to increase the congestion window. In a gigabit network this can take more than an hour although it take much less time in network with lower bandwidth. Therefore researchers are investigating new methods.

The 3rd goal of DataTAG is to find methods for Grid interoperability. There exist other Grid projects and DataTAG tries to make different Grid networks work together.

The Members of DataTAG are research groups from different countries in Europe:

- CERN (Switzerland)
- INFN (Italy)
- INRIA (France)
- PPARC (UK)
- University of Amsterdam (The Netherlands)

It is important to notice that the work that is described in this report emerged from the fact that there were so many people located at different places in Europe and the U.S. using the DataTAG testbed link. Details on that can be read in the following sections. This project was about the development of a centralized reservation application to manage the usage of the transatlantic gigabit network link provided by the DataTAG project. This report should be of interest to people who are interested in a solution to manage reservations in a distributed environment using freely available tools and software or to people who would like to continue to develop the “Reservation Application for a Transatlantic Gigabit Testbed” at CERN. It will further be discussed what the encountered problems were and how they were worked out with freely available tools and programming skills.

This work was done in the context of an M. S. thesis for The Swiss Federal Institute of Technology during 6 months. The tools and the environment were provided by CERN (European Organization for Nuclear Research) in Switzerland.

The members of DataTAG project tried to improve performance and quality of service in high-speed networking for Grids. The testbed consists mainly of a 2.5 Gigabit Optical Ethernet link located between Geneva and Chicago. At both ends two routers connect 6 Linux PCs from where the data is generated and sent through the link or received from the link. As operating system Linux was used to allow users to modify the kernel and change configurations of the machines easily modifying source. Therefore its use was straightforward since members of DataTAG tried to improve protocols at different layers in the network and the testbed was mainly used to measure performance of their algorithms.

The reservation application described in this report facilitated the management of such a link. It went through several development processes where the functionalities were refined and new ones added during the whole project. New features had to be added requested by members of DataTAG. Because of the ongoing changes of the requirements the initial architecture had to be changed again which was not planned at the beginning.

The whole application was developed from scratch. It touched many interesting parts from several domains like Unix Administration, Internet software development and IT security. The reservation application turned out to be useful to many users of the link. A lot of positive feedback pushed the development to be continued and new features to be added. At the final stage of development users from other research labs (e.g., from Manchester University) asked for the source code to use it also for different sites.

2.4 Outline

The following part of the report will go into more details about the work that has been done within this M. S. thesis. The whole project was related to the transatlantic testbed provided by DataTAG. In this report a top-down structure has been followed, starting from the very general and going more and more into details.

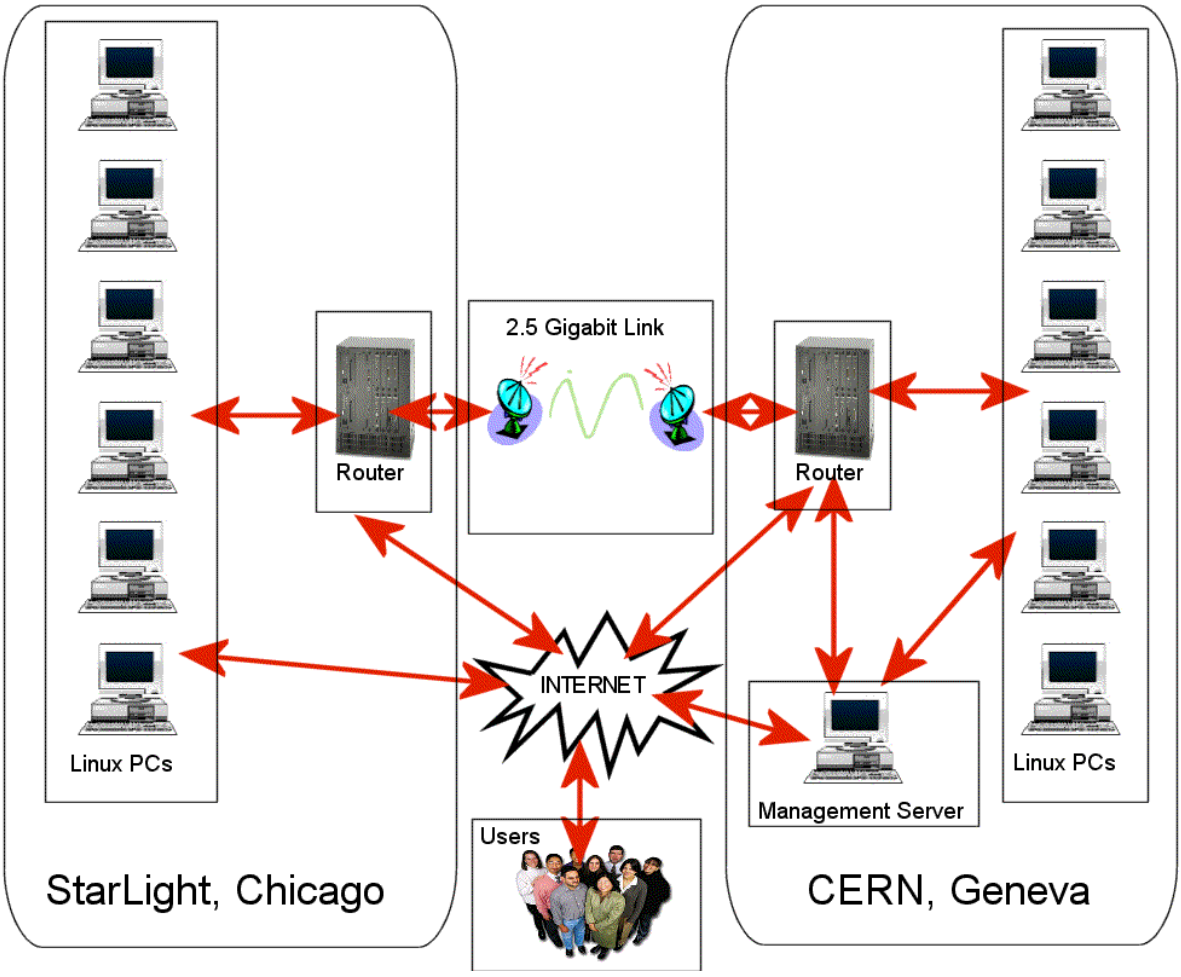
The next chapter starts with an overview of the situation. The transatlantic testbed and its hardware will be explained, to understand the problems that have occurred and to analyze the requirements to solve the emerging problems.

What will follow is the general description and architecture of the solution. Afterwards its high level design will be illustrated to justify the selection of some protocols used. Finally some of the most important technical problems that had to be solved will also be mentioned. The report concludes with a summary and plans for future work.

3 Transatlantic Testbed and its Challenges

3.1 Situation description

The following picture illustrates the situation of the hardware that was used in the transatlantic Gigabit Ethernet link:



First of all notice that there are two big boxes, the one on the left corresponded to the site in Chicago in the United States and the one on the right to Geneva in Switzerland. At both locations 6 Linux PCs were installed. Each of them contained a dual motherboard with 2.2 GHz dual Xeon processors in Chicago and 2.4 GHz dual Xeon processors in Geneva. They also had 3 network interface cards. Two of them were optical 1-gigabit Ethernet cards and the third one was a FAST Ethernet card (100 Megabit/s). The optical cards were connected to the router at each side and the FAST Ethernet card was connected to the Internet such that the users could remotely

connect to the PCs. The purpose of the optical cards was to generate a high amount of traffic to saturate the 2.5-gigabit Ethernet link, which connected the site in Chicago to the one in Geneva.

The final box that had not been discussed yet was the one that symbolizes the users of the link. The users were the DataTAG project and researchers from the Californian Institute of Technology and SLAC. These users were not located at one site but dispersed all over the world and living in different timezones.

3.2 Interferences using the Transatlantic Testbed

The routers forwarded traffic from any of the connected PCs to the link. Users could therefore interfere while either logged into the same machine or one of its neighbors. A possible scenario would have been when a user launched client-server programs and another one without awareness of what was already going on set up a new kernel and rebooted the machines. The first user would have lost all the data of his measurements that he wanted to acquire. In another situation two different users could send data through the link at the same time from different PCs, which would have affected performance measurements in both experiments.

Several more examples could be enumerated. To improve the situation a management system was needed. Using email to reserve the link proved to be inefficient. There were many mailing lists and not every user read his mails before accessing the link. That was why the DataTAG researchers came to the conclusion that measurements made before having the reservation application were meaningless.

To give an estimate the number of users in the United States is about 10 and the number of users in Europe about 20 people.

Some of them accessed the link on a daily basis.

Three initial requirements could be enumerated:

3.2.1 Allow users to perform or cancel reservations

It was necessary to have an interface that could be used by DataTAG users around the world to generate reservations, check the reservation state of the link or to delete reservations.

3.2.2 Display any scheduled reservations

It was required to see who was planning to use the link the reservations had to be retrieved by the client, enabling users of the link to schedule their own reservations or simply to verify if the link was used at the moment.

3.2.3 Manage access to the link

The third requirement was enable access to the link to the right users and disable it to the wrong ones. Experiments could so be launched without being disturbed by others.

The solution described in the next section was trying to satisfy the initial requirements. It turned out during the project that the requirements changed and modifications had to be made.

3.3 The Architecture of the Reservation Application

This part will talk about the basic concept of the reservation application. The following picture illustrates how the different components worked together.

The reservation application was based on a three-tier application system.

3.3.1 The First Tier

The first tier was the client, which was used to perform a reservation or display reservations from any point in the Internet using a Java plug-in enabled browser. The client corresponded to the user box in the picture, which describes the testbed. With Web-browsers users accessed the reservation application server from anywhere.

3.3.2 The Server Tier

The second tier corresponded to the server running Linux. The server used a Java Servlets container, which stored all reservations and reconfigured the Linux PCs connected to the DataTAG. The “Management Server” box in the testbed description picture above represents it.

3.3.3 The Linux PC and Router Tier

The third tier corresponded to the Linux PCs that had to be managed by the reservation application server and also to the routers that forwarded packets from the PCs to the link. Whenever a reservation was due the reservation application server had to lock or unlock the reserved PCs and to shut down or restart interfaces from the routers.

To communicate between all tiers some standard protocols and software were used. In the first versions of the reservation application the client and the server communicated over HTTP; in the last version, they used HTTPS when authentication was added. The reservation server used SSH to communicate with the Linux PCs to lock or unlock them through modified configuration files.

3.4 Design choices

3.4.1 Why using HTTPS (Secure Hyper Text Transfer Protocol)

HTTP was the most used protocol in the Internet for surfing the Web. Therefore it was implemented in many browsers and also Web servers used it to communicate. For this reason this protocol was also allowed to pass through many firewalls.

HTTPS is the secure version of HTTP using SSL (Secure Sockets Layer). It could add the features of client- and or server-authentication and also encryption and decryption of all sent and received data. In the reservation application only the

second feature was used. Client authentication was managed differently using passwords. Another important point was that HTTP and HTTPS were supported by Java and thus used in the reservation application.

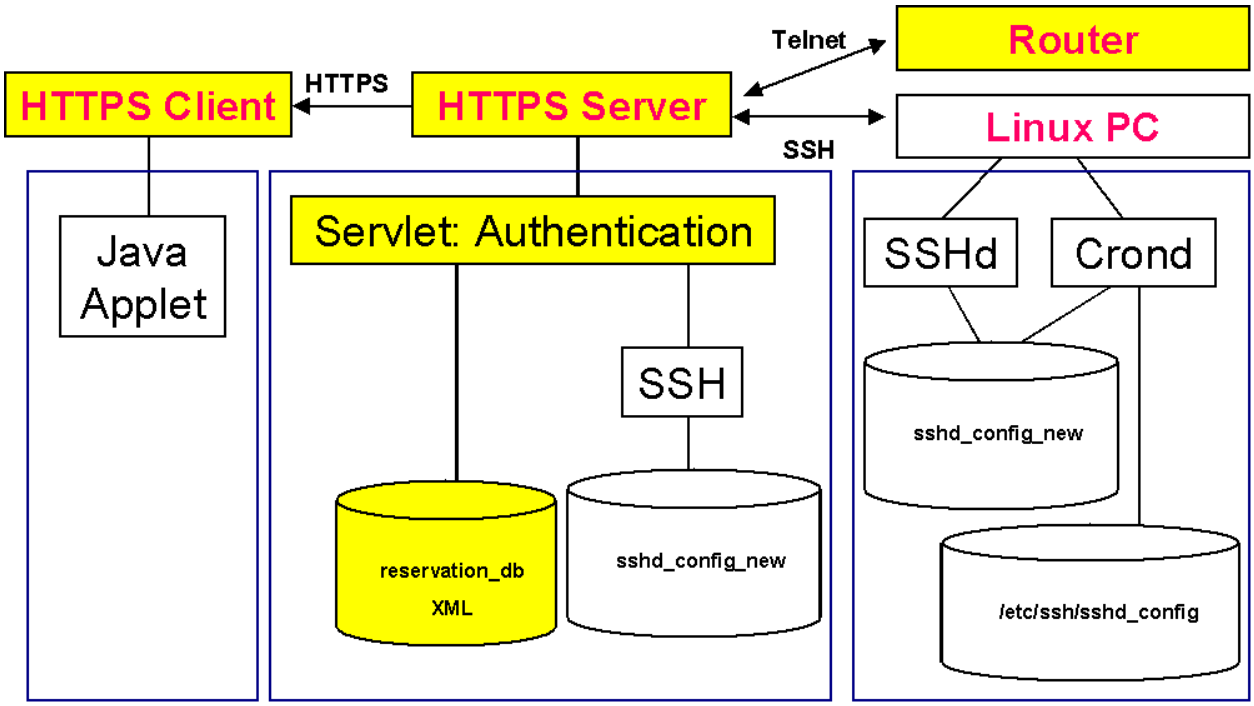
3.4.2 Advantages of SSH (Secure Shell)

SSH was a secure version of Telnet. It shipped with every modern Linux distribution. It was secure in the sense of enabling client and server authentication and also confidentiality as in SSL. Some very convenient features were that it had several types of authentication such as password or public key authentication. Public keys could be used to execute remotely scripts on Linux PCs from the reservation application without the need of providing a password manually. This was also the reason why it had been used in the reservation application. The private keys were taken from a local directory on the client PC whenever SSH had to authenticate, instead of any typed-in password.

SSH also provided methods to send or receive files to or from other PCs using its tunneling features. Tunneling in this case meant that SSH opened a connection to the server and used its cipher streams to send or receive files. There were two such services: one was called “Secure Copy” and the other was SFTP (Secure File Transfer Protocol). ”Secure Copy” was used to send the configuration files to the Linux PCs for the management process of the Linux PCs.

3.5 The Reservation Application in more detail

The following picture illustrates the reservation application with all its tiers.



When a user wanted to use the reservation application what he needed to do was to launch a Java plug-in enabled browser and to load the reservation application client. From that point he could authenticate himself using his password already used for other purposes.

On the HTTPS server a Java Servlet container was installed that could reply to the Java Applet client. To authenticate the user, the server's standard shadow file on was used. Since people used the same passwords already to authenticate to the DataTAG routers, this was a convenient way that needed only one password per user to be remembered. Users connected to the routers using an authentication scheme from a RADIUS server, which was the same server, where the reservation application server was running. The Radius system and the reservation application used the same user passwords. That was why one and the same shadow file could be used. The shadow file was the location where all user passwords were stored in the Linux operating system.

If the authentication was successful a session-id was generated using the API from the Servlet and sent to the client. From that moment this session-id was used as

reference of the server's state and sent by the client with every query. For all the queries the server looked up data in its reservation storage. The storage was an XML-file that was read from disk once the server had started and it was written back after each modification, creation or cancellation of a reservation.

During the startup, while the reservation XML-file was read, or when the client created a new reservation, the server checked the end time of the reservation. If the end time of the reservation were pointing to future, the reservation was scheduled in the server's task-scheduler. The task-scheduler, whether directly or when the start time was due, executed a shell script with some arguments that were stored in the reservation. This script generated configuration files from the arguments and sent them to the reserved Linux PCs using Secure Copy (SCP). Any Linux PC would periodically check if there had arrived a new configuration file. If this was the case it would have copied the new file to the appropriate place and asked the SSH server daemon of the local PC to apply the configuration file. SSH was the only access point for users to the PCs.

3.6 Technical Concepts of the Client

There were several versions of the reservation application. The first two versions only included a binary storage file to manage persistence. They did not have any kind of authentication. Users could make reservation with no time overlap (mutual exclusion). They also only could reserve for a single user id. This implied that all the Linux PCs only allowed access to one user during a reservation.

The last version included authentication and multiple user reservation.

3.6.1 Cronjobs and SSH

This section will explain how the link was managed by the reservation application.

Users basically connected to the PCs to access the link implicitly. SSH has become the standard that replaced Telnet to connect to Linux PCs. The most important issue with Telnet was that the data between the Telnet-client and the Telnet-server was not encrypted and logon passwords could be easely “sniffed”. SSH used modern cryptographic primitives. It encrypted and decrypted all the traffic also during the authentication procedure. The configuration features of SSH were also very powerful. In the SSH server configuration file one could specify to grant access to some specific users and lock other users out. This was the basic feature used to implicitly prevent people from sending data through the link. If they could access any PC they could not send traffic through the link.

The other service, which was needed, was the Cronjob service. This service allowed scheduling of tasks that were executed periodically. In the case of the reservation application it was used at the server in the beginning and at the PCs until the end of the project. The server launched periodically a “bash” shell-script, which checked through a C program if a reservation was about to start or about to end. If the state of a reservation changed, it would have generated a new configuration file for the PCs and have sent it using Secure Copy (SCP). The Cronjob service at the server was replaced with a Java scheduler specially developed for the reservation task application.

The Cronjob service was also running on all Linux PCs. It was used to check if a new SSH configuration file had arrived from the management server and then to accordingly reconfigure the local SSH server daemon.

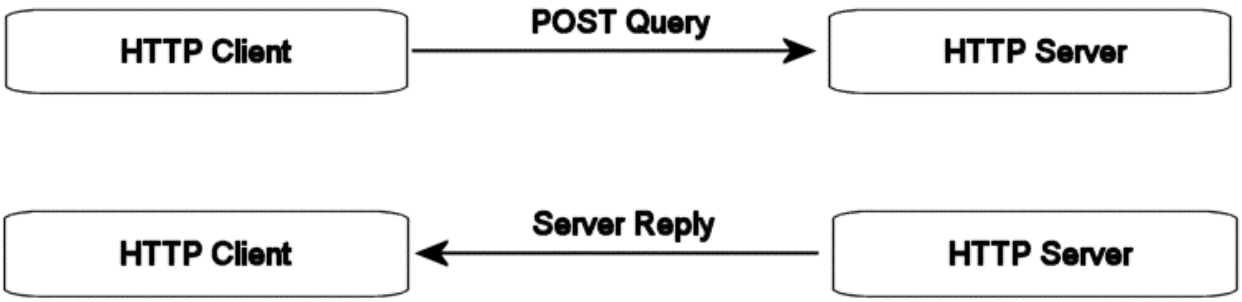
3.6.2 Using Java Applet with SUN's standard plug-in

The first question that came up was: How should users be able to perform reservations? What kind of interface would be appropriate on what systems? Since people could not use the link locally from one point, the World Wide Web had to be used. The software used to connect to the server had to be something CERN already owned or which was free to use and that users could run from distance. Therefore it was decided to use browsers to perform reservations. Browsers use GET and POST methods to send data.

3.6.3 GET and POST to communicate between client and server

In this section it will be explained what methods were used to communicate and how they could be used by client and server programs. Both methods were specified in the RFC 2616 document from IETF, which defines the HTTP/1.1 protocol.

The GET method was used to request data from a Web server. Any browser that connected to a HTTP server to display its content used a GET query. The requested data was specified in the URI (Uniform Resource Identifier). Therefore data sent to the server were encoded in the URI. Although HTTP/1.1 did not limit the size of the URI in its specifications, all servers and browsers do. Some Internet Explorer versions enforced the size of an URI to be at most 2083 characters. Therefore GET was not suitable to send large amounts of data.



The other variant to send data was POST. It also used a URI but only to specify the context at the server. The data was not concatenated to the URI. With this method the length of the data was sent separately and was therefore not limited.

Browsers for example were able use both methods within HTML forms to send the form to the Web server.

3.6.4 HTML based user interface was not enough

But some client-side calculations had to be considered as well. For example time zone management was an important problem. Since some users were located in the United States or United Kingdom it was important to reserve the link at the right unique time.

There would probably be a way to obtain the time zone used by the client checking Meta-data from the GET or POST request at the server. But what if someone used a proxy that was not located in the same time zone?

For this reason, and because of the power and flexibility Java gives to create a graphical user interface, it has been decided to use Java Applets as client interface. Some well-known browsers such as Netscape, Mozilla, Opera and Internet Explorer also support Java.

3.6.5 Incompatibility between different browsers with Java

A new problem occurred once the first applet has been developed because of different virtual machines that ship with different browsers. The API was not the same on all virtual machines and therefore the applet did not behave the same way on all browsers. SUN has therefore provided a plug-in for the most common browsers. With special HTML-tags browsers are forced to use the virtual machine from SUN's plug-in instead of the browsers virtual machine. Those HTML-tags are again different on Netscape and Internet Explorer. SUN therefore provides an HTML-converter, which adds Java-Script to the HTML-file containing the basic Applet-tags to select the Java plug-in at the browser. This HTML-converter ships with every Java Development Kit (JDK) v1.3 or above.

3.6.6 HTTPS using Java plug-in

And finally there was something more very useful with the Java plug-in. It was possible to use the HTTPS support from the browser. There were several problems one could encounter using SSL or HTTPS with Java.

Before JDK 1.4, the “Java Cryptography Extension” was not bundled with the JDK: It had to be downloaded separately and installed afterwards.

An HTTPS server needed to have a certificate installed, which contained its public key. When a client requested data from an HTTPS enabled server it received the certificate during a handshake procedure from the server. This certificate must have been signed by at least one party or it could have been self-signed. The client may have trusted and accepted the certificate or aborted the connection.

Self-signed certificates were not trusted by default in the “Java Cryptography Extension”. Java would have thrown an exception indicating that the certificate was not trusted if “VeriSign” or “Thawte” had not signed it. There were three ways to solve this problem if one did not want to buy a signed certificate from a trusted company. One was to manually install the self-signed certificate in the Java runtime environment. Another possibility was to write a new trust-manager that accepted self-signed certificates in the client itself. Finally, if one used the Java plug-in within a browser, it could use HTTPS capabilities from the browser to connect to the server. Browsers popped up a window if someone tried to connect to a HTTPS server with a certificate, which was not signed by a trusted authority, listed in the browser. Users could then decide by themselves if they wanted to trust the certificate by verifying information about the issuer in the pop-up window. The Java plug-in applied the policies selected in the pop-up window. Therefore it was very convenient to use the Java plug-in with HTTPS.

Java plug-in enabled HTTP and HTTPS communication through the “URL”-class.

3.7 Technical Concepts of the Server

3.7.1 Using Common Gateway Interface (CGI)

In the first 2 versions of the reservation application CGI had been used. Since the concept of communication is the same using CGI and Java Servlets, both of them use POST or GET requests. Both CGI and Servlets were used to develop the reservation application.

POST and GET methods have been discussed above from the client point of view. The server could react to those methods using different methods. Common Gateway Interface was an older standard to generate dynamic web pages. It could handle POST and GET methods and retrieve information through environment variables. Sending data to a CGI was performed by writing to or reading from local “Standard Input/Output”-streams.

To obtain the data that was sent by the client through a GET request the “QUERY_STRING” environment variable had to be decoded, which was generated by the HTTP server and passed to the CGI. The CGI was an execution of a process that had access to the environment variables and to Standard Input/Output streams. Thus, almost every language could be used.

To reply to a request the CGI simply had to writes the requested data to the Standard Output stream.

To retrieve the data sent by POST-methods the CGI read the data from Standard Input instead of an environment variable. This was also the reason why the length of data sent with POST was not limited like with GET. Standard Input/Output were streams. The “QUERY_STRING” environment variable was limited by the size of an URI that could be specified at the browser and the size of the URI than could be accepted by the HTTP server.

3.7.2 Reservation Management Storage

The reservation storage consisted of a file. In the first two versions of the reservation application, a binary file was generated to store the reservations. Afterwards the binary file turned out to be not flexible enough to add new fields and a new requirement asked it to be human readable such that it could modified using text editors.

3.7.3 Binary file storage

To keep the file structure simple a binary file was created where offsets in the file corresponded to the start time and end time of a reservation. The values at a given offsets identified the user that had reserved the link for the corresponding time. If the value was zero it meant that there was no reservation made. The first offset of the reservation file represented a start date of all possible reservations. No reservations could be performed before that date since it corresponded to the first offset. The last offset indicated the end date of every possible reservation. The reservation file could not store dates after the end date.

For example, let us suppose that the binary byte string “0 0 1 1 0 0 0 0 2 2 2 2” has to be interpreted. Let us assume that it was a sub-string of the whole binary file.

Let us say that one byte corresponded to 15 minutes. The two “zero” bytes indicated that there was no reservation made for 30 minutes followed by a reservation lasting for 30 minutes by user1. The next 4 zeroes indicated again a time slot of 1 hour where the link was not used followed by a reservation from user 2 for at least 1 hour.

This structure had advantages and disadvantages. A plus was that it was simple to use and that the reservations could not overlap since it was only possible to have one user at a given offset. In addition, the size of the file was small and constant. To display all the reservations of a given day was also easy. It was enough to sequentially read the file from the starting offset of that day and stop the reading before the offset of the next day.

To perform a reservation the client simply sent the offsets corresponding to the reservation time and a user identification code to the server.

One disadvantage was that the file could only be used for a given amount of time since its size was constant. In addition, adding new features to the reservation like the possibility to reserve for multiple users was not very comfortable with this file structure. The whole storage part of the source code had to be changed at the server and at the client. That is why this kind of file structure was used only with the CGI architecture. Later, when Java Servlets were used the file structure was changed into a human readable one based on XML (Extensible Markup Language).

3.7.4 Migrating to Java Servlet architecture

Since the requirements changed a Java Servlet had to replace the CGI executable that was written in C. The former storage structure proved to be inefficient. New features could not be easily added. And the file was not human readable in case of a failure. The Cronjob service at the server was removed and replaced with a scheduler based on Java-Threads.

3.7.5 Replacing CGI with Java Servlet

As already described CGI programs used Standard Input/Output to communicate. The Java servlet class worked quite similarly. It provided methods for GET-requests and POST-requests. Both methods had two objects as arguments that corresponded to Standard Input/Output. There was one object corresponding to the request and another one to the response. The request object provides an input stream from where one could retrieve data sent with the POST request from the client. To reply to the client the servlet class provides a Response object that contains an output stream. This stream was used to send the data back.

3.7.6 Changing the structure of the reservation storage file

Changing the structure of the reservation file from a binary file into a human readable was a little bit more complicated. In the binary file it was easy to verify that only one reservation was made at a time such that no overlaps occurred between different reservations. And also the retrieval of reservations was easy given the special structure of the file. Using a text-file needed more processing to verify that two reservations did not overlap. The reservations also had to be sorted by time when they were inserted for queries, which they were implicitly in the binary file. Thus a sorted linked list was a good way to store the reservations in memory. Queries of sequential reservations became again easy, for example to retrieve the reservations made for a given day.

3.7.7 XML storage file linked list

XML has become standard for storing data in a text file. It is similar to HTML but has some stricter rules concerning the structure. Tags are only used to describe data and not for formatting purposes.

Java supports XML using the Java package called Xerxes. This package is included since JDK version 1.4 and it has to be added in all earlier versions of JDK. There were also other packages related to XML available in Java but they were not needed for this reservation application. Xerxes basically provides a library for two kinds of parsers. One is based on DOM (Document Object Model) the other is based on SAX (Simple API for XML). In the reservation application, DOM was used although SAX would have worked equally well. DOM basically generated trees from XML files. These trees could be traversed and the data interpreted while traversing nodes. In the reservation application the linked list was generated at the start of the server while traversing the XML-tree. Every time a new reservation was made it was inserted into the linked list and stored in the XML file. If a reservation was to be deleted it would first be removed from the linked list and then an XML file would be written. What has been gained by doing that?

It had become easy to store data. One could add data by simply appending tags. It was also possible to modify the file with a text editor.

But verifying the integrity of the data had become more complicated. In the first two versions of the “Reservation Management System” the reservations made by users were stored in the binary file. Time overlap of different reservations was not possible.

Here an example of the reservation XML-file containing one reservation:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ReservationList SYSTEM "datatag_reservation.dtd">
<ReservationList>
	<Creation>
		<TimeInfo>
			<Time>16:30</Time>
			<Date>03.02.2003</Date>
			<Zone>GMT+01:00</Zone>
		</TimeInfo>
	</Creation>
	<Update>
		<TimeInfo>
			<Time>16:14</Time>
			<Date>24.02.2003</Date>
			<Zone>GMT+01:00</Zone>
		</TimeInfo>
	</Update>
	<Reservation>
		<UserList>
			<User>emartell</User>
			<User>sravot</User>
		</UserList>
		<Start>
			<TimeInfo>
				<Time>09:00</Time>
				<Date>25.02.2003</Date>
				<Zone>GMT+01:00</Zone>
			</TimeInfo>
		</Start>
		<End>
			<TimeInfo>
				<Time>10:55</Time>
				<Date>25.02.2003</Date>
				<Zone>GMT+01:00</Zone>
			</TimeInfo>
		</End>
		<CernList>
			<IP>w01gva.datatag.org</IP>
			<IP>w02gva.datatag.org</IP>
		</CernList>
		<CaltechList>
			<IP>w03chi.datatag.org</IP>
			<IP>w04chi.datatag.org</IP>
		</CaltechList>
		<BootKernel>default</BootKernel>
	</Reservation>
<ReservationList>
```

In this example one can see how a reservation was structured in the XML storage file. It had tags to identify the users that reserved the link. Also the start and end times of a reservation were stored, followed by a list containing the reserved Linux PCs. The “Boot-kernel” tag was planned to enable reboots of requested

kernels at the beginning of each reservation. It was not used in the end because the different Linux PCs did not have the same configuration.

3.7.8 Linux “Shadow” authentication

The third version of the reservation application included authentication through shared secret, i.e. standard user passwords used at a particular Linux PC. This PC was also used for the reservation application and as a RADIUS-server to authenticate users at both DataTAG routers connected to the link. Password management was simple. Users only needed to remember one password to use the reservation application and to connect to the DataTAG routers.

Linux used a file located in the /etc folder which is called “shadow” to store its passwords. This text file contained information about passwords and their validity from every user that has account on that PC. Passwords were stored in an encrypted manner. Every time a user wanted to authenticate, his typed-in password would be encrypted using a specific algorithm and then compared to the entry in that file. Only encrypted passwords were compared to the entry in the shadow file to protect passwords from system administrators.

The following picture displays a sample entry of the user “sreto” in a “Shadow”-file:

```
sreto:$1$02TYebfH$222TLx2DKePr55oJSheE.0:12024:0:99999:7:::
```

The colons are separators between the different fields. “sreto” identified the user. The sequence of characters after the separator (`$1$02TyebfH$222TLx2DKePr55oJSheE`) represented the encrypted password of user “sreto”. One could split this sequence into three parts separated by the “$”-character. The first one identified the version of the encryption scheme that was used. In this case it showed that MD5 hash functions were used to encrypt the password. The second sequence corresponded to the “salt”. Salts were used to make sure that users having the same passwords on the same machine had the

same representation of their passwords only with low probability. And the last sequence was the encrypted password.

The problem now was to integrate this authentication scheme into the reservation application using Java. The easiest way was to write code to encrypt passwords the same way they were stored in the Shadow file.

Two encryption schemes were used in Linux. One of them, now obsolete, was based on a function called “crypt” and used Data Encryption Standard (DES) to encrypt passwords. The new one that was still used applied MD5 hash function to encrypt passwords. This was good news since Java had already implemented MD5 hash function. All the reservation application needed to do was to read a copy of the Shadow file, parse the file, generate an encrypted version from the plain password entered by the user and compare it to the parsed data. If both encrypted passwords matched, then the authentication process was successful.

A Cronjob service checked if the Shadow file had changed and made a copy into the user account folder where the reservation application was started. It also modified the ownership of the file and set restricted access rules to the copied file such that only root or the reservation account user could access the file. Thus the reservation application did not need to run with super-user privileges.

3.7.9 Concept of Sessions

The most important primitive to understand to implement authentication was the concept of session-ids. They were widely used by many Web applications and had become a standard.

Web applications used HTTP or HTTPS to communicate between client and server. The problem was that those connections were stateless by default. After each GET or POST query the connections were closed. HTTP by itself did not provide means to store the state of any transaction between client and server. Thus the concept of sessions was brought into life where the state could be stored within a session. A session-id was a randomly generated number. It had to be difficult for others to guess this number and unique to distinguish between different users and their sessions. Sessions also had a time stamp that was refreshed after each query.

The server deleted session-ids that were not used for a certain amount of time to free resources and also for security reasons.

Once a session-id was created it had to be transmitted to the client. The client used this session-id and attached it to the query. The server was then able to identify the context of the query.

The Servlet container used in the reservation application supported session-ids. The only state to store was the information if a user had authenticated himself or not. Some other information like the clients IP-address was also stored to prevent people from session hijacking. This was achieved by checking after each query if the client provided any session-id and if the provided session-id was still valid.

4 Conclusion

4.1 Summary

This report described the successful development of a distributed application that manages one resource. It solved problems where users of the transatlantic testbed interfered with each other using one single resource from different locations. Since the users of the testbed were scattered over two continents, working in different timezones, they could not simply announce the use of the resource by talking to other users. Before the existence of this reservation application, the users of the transatlantic testbed had nothing but a mailing list as a tool to avoid interferences.

The software was developed in multiple iterations. New features were added with every version and used immediately by users.

The goal of this project was not only to provide a solution at the end of the internship but also during its development phases.

Finally a researcher from the University of Manchester, who was also working in DataTAG, has shown interest in the source code of the reservation application. He wanted to use it in another environment not related to DataTAG.

Researchers from Amsterdam wanted to use part of the source code in their Grid application to be able to reserve the link without the client Applet.

4.2 What did I learn

I had to work under real-world constraints that challenged me to develop solutions that could be quickly used.

The reservation application was developed from scratch using freely available tools such as Java, Tomcat Servlet Container, Apache, SSH and Shell-scripting. I had to go through all elements of a development process such as analysis, design, testing and multiple interactions.

To achieve the goals of this projects many concepts had to be learned and to be correctly implemented. The first encountered problem was to analyze the situation and provide quickly a solution. Many of the tools used in the project were new to me.

I have learned concepts of Web-based application programming and Linux administration.

Other issues were the ongoing changes of user requirements that occurred during the development process. I have learned to think of more flexible solutions already from the beginning. Reorganizing my application to adapt to the new requirements took a lot of time.

Finally I have improved my knowledge of distributed application development, of using advanced Java features and of writing Shell scripts in Unix environments.

4.3 Future work

At the end of the last release more work needed to be done. Some of the Linux PCs at CERN were not only used for the DataTAG transatlantic link but also accessed across GEANT, the European R&D network. Thus a new feature was requested that enabled reservation of Linux PCs without reserving the DataTAG link. In the current version, it was only possible to reserve PCs including the DataTAG link.

Since some other members of DataTAG have requested the source code it turned out to be useful to shrink-wrap the whole software package to facilitate deployment in other environments.

5 Acknowledgements

I wanted to thank J.P. Matin-Flatin, Technical Manager of the DataTAG Project, who arranged my internship at CERN and supervised my project at CERN during the 6 months. He has given me many good advices to help me to get through the whole internship.

My thanks also go to Prof. Schiper from EPFL, who kindly accepted to supervise my M.S. Degree final project and has also given me the possibility to explain my work to members of his research laboratory.

I also thank the whole IT-CS-EN section at CERN, where I was part of, for their support, especially to Olivier Martin who was the responsible manager of the group.

Thanks also go to Sylvain and Edoardo who helped with some technical problems.

6 Appendix

The source code can is available at:

- http://datatag.web.cern.ch/datatag/documents/reservation_application_source_v3_27_02_2003.zip

7 References

[1] Olivier Martin, Technical Annex 1 Amendment 1 of DataTAG project (IST-2001-32455), 2.April 2002

[2] John December, Mark Ginsburg, HTML & CGI UNLEASHED, Sams.net Publishing, 1995

Links related to Open Source Web servers and Servlet containers:

- http://httpd.apache.org/
- http://jakarta.apache.org/
- http://www.johnturner.com/howto/apache-tomcat-howto.html
- http://www.apl.jhu.edu/~hall/java/Servlet-Tutorial/

Links related to Java programming:

- http://developer.java.sun.com/developer/codesamples/examplets/
- http://www.javaworld.com/javaworld/javatips/jw-javatip96.html

Links related to HTTPS with Java:

- http://java.sun.com/products/plugin/1.2/docs/https.html

Links related to XML with Java:

- http://members.fortunecity.com/seagull98/XmlTutorial.html

Links related to Java Swing programming:

- http://java.sun.com/docs/books/tutorial/uiswing/
- http://www2.gol.com/users/tame/swing/examples/SwingExamples.html

Links related to Shell programming:

- ftp://www6.software.ibm.com/software/developer/library/bash.pdf