A differentiated quality of service oriented multimedia multicast protocol

David Rafael Garduno Barrera Garduno Barrera

Thèse

Préparée au Laboratoire d'Analyse et d'Architecture des Systèmes du CNRS

En vue de l'obtention du Doctorat de l'Institut National Polytechnique de Toulouse

Ecole doctorale : Informatique et Télécommunications

Spécialité : Réseaux et Télécommunications

Par **GARDUNO BARRERA David Rafael**

A DIFFERENTIATED QUALITY OF SERVICE ORIENTED MULTIMEDIA MULTICAST PROTOCOL

UN PROTOCOLE MULTIMEDIA MULTIPOINT A QUALITE DE SERVICE DIFFERENCIEE

Soutenue le 8 Avril 2005 devant le jury :

PANSIOT Jean-Jacques

BOUABDALLAH Abdelmadjid

DIAZ Michel

FRABOUL Christian

GAYRAUD Thierry

MATHY Laurent

LEBLANC Philippe

Acknowledgements

This work was conducted at the “Laboratoire d’Analyse et d’Architecture des Systèmes (LAAS/CNRS)”, Toulouse, France, in the group “Outils et Logiciels pour les Communications (OLC)”, and partially supported by the “Consejo Nacional de Ciencia y Tecnología (CONACyT)”, México. I thank both institutions for their confidence and support.

I would also like to thank Mr. Jean-Claude Laprie and Mr. Malik Ghallab, successive heads at LAAS, for having received me in this laboratory and for having given me the opportunity of successfully performing this work. I thank also Mr. Michel Diaz and Mr. Jean-Pierre Courtiat, successive heads of the OLC research group for their direction, assistance and support.

I express my deep gratitude to Mr. Michel Diaz, my research director, for his priceless guidance and assistance, for his support and confidence in this research work. I thank also Mr. Thierry Gayraud for his precious help and support and for his invaluable assistance in this dissertation.

I thank Mr. Jean-Jacques Pansiot and Mr. Abdelmadjid Bouabdallah for having accepted the hard task of reviewing and evaluating this work and for their helpful notes and comments which have helped me to enhance it. I thank also Mr. Christian Fraboul, Mr. Laurent Mathy and Mr. Philippe Leblanc for having accepted to be part of my committee, for their comments and for the interest they have shown to this work.

Along the time passed at LAAS, I appreciated the friendship of some persons that have supported me in critical moments. I would like to thank my colleagues: Guillermo Hoyos, Roberta Lima Gomes, Magnos Martinello, Valentim Dos Santos, Ernesto Exposito, Florin Racarau, Christophe Chassot and the rest of colleagues within the OLC group. I thank also the technical and administrative personal at LAAS without whom this work would not have been possible.

In this time passed at Toulouse I have known some people that have made easier my life in this country, far from my family. I have specially appreciated their friendship, their happiness and their day to day support. I thank specially Marcos Dos Santos, Elisa Urrestarazu, David, Hervas, Vicky and Nadia Gonzalez, Susy and Memo Hoyos, Omar and Lupita Alvarado, Javier and Alejandra Scheiner, Juan Nieto, Valentin, Juanito, Mariano, Rosalba Arguelles, Angélica Sierra and Altamira.

I would like to particularly thank my parents, David Garduño and Cruz María Barrera, my brothers Manelic and Azor and my friends in México. I thank their support and encouragement, even in the distance.

Finally, I want to express my eternal gratitude to Christel Le Bellec, because she believed in me and because she has been beside me in the most difficult moments. I thank specially her patience, her company and her support.

A Differentiated QoS-oriented Multimedia Multicast Protocol

Keywords: Multicast, Multimedia, Quality of Service (QoS), M-FPTP, Application Level Multicast, UML, Hierarchized Graph, Degree-Bounded Shortest-Path-Tree.

Modern multimedia (MM) communication systems aim to provide new services such as multicast (MC) communication. But the rising of new very different MM capable devices and the growing number of clients drive to new requirements for mechanisms and protocols.

In a MM communication, there are some flows that have constraints different from others and the required QoS for each flow is not the same. Furthermore, in MC communications, all the users do not want or are not able to receive the same QoS. These constraints imply that new communication mechanisms have to take into account the user requirements in order to provide an ad hoc service to each user and to avoid wasting the network resources.

This dissertation proposes a new differentiated QoS multicast architecture, based on client/server proxies, called M-FPTP, which relays many MC LANs by single partially reliable links. This architecture provides a different QoS to each LAN depending on the users requirements. For doing so, it is also provided a network model called Hierarchized Graph (HG) which represents at the same time the network performances and the users QoS constraints. Nevertheless, the application of standard tree creation methods on an HG can lead to source overloading problems. It is then proposed a new algorithm called Degree-Bounded Shortest-Path-Tree (DgB-SPT) which solves this problem. However, the deployment of such a service needs a new protocol in order to collect users requirements and correctly deploy the proxies. This protocol is called Simple Session Protocol for QoS MC (SSP-QoM).

The proposed solutions have been modeled, verified, validated and tested by using UML 2.0 and TAU G2 CASE tool.

Un protocole Multimédia Multipoint à Qualité de Service Différentiée

Mots clés : Multipoint, Multimédia, Qualité de Service (QdS), M-FPTP, Multipoint au Niveau Application, UML, Graphe Hiérarchisé, Arbre de Plus Courts Chemins à Degré de Sortie Limité.

Les systèmes de communication multimédia modernes aspirent à fournir de nouveaux services tels que des communications multipoints. Néanmoins, l'apparition de dispositifs multimédias très diversifiés et le nombre croissant de clients ont révélé de nouveaux besoins pour les mécanismes et les protocoles.

Dans une communication multimédia, les flux présentent des contraintes différentes et la QdS requise pour chaque flux n'est pas la même. De plus, dans une communication multipoint, tous les utilisateurs ne peuvent pas ou ne sont pas capables de recevoir la même QdS ; cette contrainte implique que les nouveaux mécanismes de communication doivent prendre en

compte les besoins des utilisateurs pour fournir un service adéquat à chaque utilisateur, surtout pour éviter le gaspillage des ressources réseau.

Cette thèse propose une architecture multipoint à QdS différentiée appelée M-FPTP. Basée sur des proxies client/serveur, elle relie plusieurs LANs multipoints à travers des liens point-à-point partiellement fiables. Cette architecture fournit une QdS différente à chaque LAN dépendant des besoins des utilisateurs. Pour ce faire, nous proposons un modèle du réseau appelé Arbre Hiérarchisé (AH) qui représente en même temps les performances du réseau et les contraintes de QdS des utilisateurs. Nonobstant, l'application de méthodes standard pour la création d'arbres sur un AH peut conduire à des problèmes de surcharge du degré de sortie dans la source. Pour résoudre ce problème, nous proposons alors un nouvel algorithme appelé Arbre de Plus Courts Chemins à Degré de Sortie Limité. Le déploiement de ce service nécessite, pour gérer les utilisateurs et le déploiement correct des proxies, un nouveau protocole appelé Protocole Simple de Session pour QdS multipoint.

L'ensemble des solutions proposées a été modélisé, vérifié, validé et testé en utilisant UML 2.0 et l'outil TAU G2.

Summary

Introduction

Multimedia (MM) systems are evolving very fast and the diversification of new applications, the growing number of users and of new very different MM-capable devices have led to new requirements for Quality of Service (QoS).

In a multimedia communication, there are typically some flows that have different constraints from others and the required QoS for each flow is then different. This leads to the necessity of providing strong QoS provisioning mechanisms on a few flows while providing lower QoS for the others in order to optimize the available resources. There exist some new generation transport protocols targeting partial QoS MM transmission, as SCTP (Stream Control Transmission Protocol) [STE00], DCCP (Datagram Congestion Control Protocol) [KOH02] and FPTP (Fully Programmable Transport Protocol) [EXP03] which are based on a point-to-point architecture.

Modern MM applications are addressing not only two users, but sets of users. This group communication, also called, in a wide sense, Multicast, is the focus of an intense study in the internet research community. One possible solution to these requirements is IP multicast. Since its proposition in the late 80s and in spite of numerous efforts of a generation of researchers, many unsolved issues remain in the IP multicast model that delay its development, deployment and ubiquity. One of the prominent issues is the lack of QoS requirements on the distribution tree creation. The solutions proposed for the mono-flow point-to-point systems are insufficient for the case of multimedia multicast systems.

This dissertation aims at proposing a solution to the basic multi-user problems by proposing a differentiated QoS-oriented multimedia multicast group communication protocol.

Context

This work started in the IST GCAP (Global Communication Architecture and Protocols for new QoS services over IPv6 networks) [GCAP, GAR01, OWE01]. GCAP aimed at the development of new architectures and protocols for multimedia multicast transport protocols. One of the results of this project was FPTP [EXP03]. FPTP is a QoS oriented new generation transport protocol. It provides a set of important transport mechanisms oriented to application

requirements by using the existent network services and resources in an ordered and timed optimized way. FPTP is a configurable and programmable protocol that provides point-to-point communication services which satisfy the QoS constraints of distributed multimedia applications.

In the GCAP project, some tests were performed between a sender at *Universidad Carlos III* in Madrid and a receiver at LAAS in Toulouse by using the programmable and active platform SARA [SARA] in order to deploy FPTP proxies [DIA01, EXP02, EXP02-2]. The results were concluding and showed that traditional point-to-point multimedia connections could be enhanced by using the proposed FPTP architecture.

Contributions of this work

The purpose of this thesis is to extend and enhance the results obtained in the context of the previous projects. Its goal is the creation of a differentiated-QoS-oriented multimedia multicast tree fitting well the partial order and reliability characteristics of the FPTP protocol.

The first contribution of this work is a modified FPTP proxy architecture. The original FPTP proxy architecture permits to connect two clients placed on distant LANs through a pair of proxies. This dissertation proposes a first FPTP proxy extension aiming at interconnecting all the users in a local multicast LAN with those placed in a distant LAN by using a FPTP link (MC-to-MC). The idea of extending the point-to-point FPTP capabilities to point-to-multipoint was partially developed in the RNRT @irs++ [@irs++] project. The goal was to extend the FPTP unicast capabilities in order to create a single-source multimedia multicast communication. The result of this work was a multicast service based on a simple single level tree formed by single and independent FPTP connections. This work needed to modify the original FPTP proxy architecture. In order to relay a set of LANs, it was proposed a new modification of the proxy architecture which permits to define a different QoS for each LAN. This architecture extends the point-to-point FPTP capabilities to point-to-multipoint. Nevertheless, for doing so, it is needed to build among the proxies a multicast tree taking into account some differentiated QoS constraints.

A second contribution is a network topology model adapted to differentiated QoS constraints called Hierarchized Graph (HG) which takes into account the network capacities and the user constraints. This graph solves also the problem of network resources wasting caused by an all-to-all policy found in a complete graph. The HG is modeled by using UML 2.0 and SDL.

The third contribution of this work is a new multicast tree creation algorithm called Degree Bounded Shortest Path Tree (DgB-SPT) which solves the overloading problem found in the source when applying a simple SPT algorithm on an HG. DgB-SPT creates a degree-bounded-tree which minimizes the distance-to-the-source for all vertices. This algorithm is based on Dijkstra's one. This algorithm is also modeled by using UML 2.0 and SDL. The UML model is then tested in order to compare its performance with Dijkstra's algorithm. The tests performed on DgB-SPT permit to show that it is possible to find a good compromise among output degree and distance to the source.

Finally, in order to integrate the previous contributions, this work proposes a session protocol called Simple Session Protocol for QoS Multicast (SSP-QoM). This protocol collects the users' QoS constraints, measures distances between FPTP proxies, and creates an HG and a DgB-SPT. This protocol puts into evidence the necessity of new mechanisms to facilitate the deployment of new network solutions and technologies. The proposed protocol can use a

programmable network platform on the edge devices in order to dynamically deploy the FPTP protocol. The proposed protocol is modeled, verified, validated and tested by using UML 2.0 and the TAU G2 CASE tool.

Dissertation Structure

The remaining of this document is organized as follows. First chapter gives a survey on the related work. It gives a description of some fundamental concepts such as Multimedia, Quality of Service, Partial Order and Reliability and FPTP. Then, it gives a brief survey of multicast history, the lacks on its deployment and some of the most used multicast algorithms. It also describes some recent work on QoS multicast and gives an introduction to a new QoS-oriented network topology model named Hierarchized Graph. Finally, it shows the necessity of systems modeling, validation and verification and explains the methodology used in this work, Model Driving Architecture, and the associated language used to support this methodology, UML 2.0.

Second chapter proposes some extensions to FPTP in order to relay many local Multicast networks by using single FPTP connections. This new architecture, named M-FPTP, allows defining a different QoS for each LAN. M-FPTP needs some multicast mechanisms in order to interconnect the proxies; nevertheless it is not possible to use the traditional IP Multicast because it is not present on most of the core networks. When the network infrastructure does not support the IP Multicast routing protocols, it is possible to use a new architecture model called Application Level Multicast (ALM). By using ALM to implement multicast communication, each participating end system (FPTP proxy) is responsible to forward the received datagrams to all other ALM members in the multicast group. Nevertheless, all clients do not want to or are not able to receive the same QoS; so a model handling different multi-user QoS is needed. It is then proposed a network topology model in the form of a graph called Hierarchized Graph (HG). This HG takes into account, at the same time, the different users QoS requirements and the network performance. The HG minimizes the QoS assigned to each proxy *x* by avoiding it to forward data to any other proxy *y* desiring to receive a QoS higher than the one provided to its own clients (x's clients).

In a MM communication, the most important network parameter to be optimized is the end-to-end delay. In graph theory, and in the context of a multicast session, this property optimization can be viewed as finding a shortest path tree (SPT) between the network elements. In chapter 3 it is shown that the application of standard algorithms in order to create an SPT based on a HG can lead to overloading problems for the source. Then, it is proposed a multicast algorithm aiming at solving this problem. This algorithm, mentioned previously, is called DgB-SPT. DgB-SPT algorithm modifies the Dijkstra's one by first adding vertices dynamically, and then constraining the maximal output degree on each vertex in the graph. The algorithm is modeled by using UML 2.0 and then validated by simulation. The obtained results are compared with Dijkstra algorithm.

The ALM architecture implies that all nodes in the session can be dynamically programmed in order to perform an extended behavior. This goal can be reached by using programmable nodes. This technology permits to economize network resources by freeing the unused nodes. The dynamic deployment of the modified FPTP proxies, by following a DgB-SPT structure, needs an adapted session protocol. Chapter 4 shows that most existing session protocols do not take into account the dynamism and the user QoS requirements and constraints. The previously mentioned SSP-QoM protocol is used to integrate M-FPTP, HG and DgB-SPT together. This protocol receives the MM clients' login requests and their QoS requirements and creates a HG and a DgB-SPT; it then dynamically deploys the modified FPTP proxies by following the tree

structure. Finally, this protocol is modeled by using UML 2.0 and the TAU G2 CASE tool and is validated by simulating the UML models.

A conclusion, together with further possible work, will terminate the thesis.

Chapter 1

Related Work

This chapter gives a brief survey of the work related to the contents of this dissertation. Section 1.1 presents a description of some fundamental concepts such as Multimedia, Quality of Service and Partial Order Quality of Service. Section 1.2 gives an outline of a new generation multimedia transport protocol called Fully Programmable Transport Protocol, its conception principles, its QoS contextual model and mechanisms, its specification and its evaluation. Next, section 1.3 shows a brief survey of IP multicast history and the lacks on its deployment. It also gives a description of some of the most used multicast algorithms and it is shown that the users' QoS requirements are not taken into account by them. In addition, this section describes some recent work on QoS multicast and gives an introduction to a new QoS oriented multicast model named Hierarchized Graph. Then, section 1.4 exposes a dynamic protocol-deployment mechanism called programmable networks and it describes two programmable platforms: SARA and JavaProxy. Finally, section 1.5 shows the necessity of systems modeling, validation and verification and explains the methodology used in this work MDA, and its associated language used to support this methodology, UML.

1.1. Multimedia, Quality of Service and Point-to-Point protocols

1.1.1. Multimedia

A *Media* is the way the information is perceived, expressed, digitalized or transmitted; while *Multimedia* refers to simultaneous and integrated utilization of different medias (ex. text, video, audio, still images or animations, etc.).

Multimedia applications provide the needed functions in order to process all information coming from users, and at the same time, the capture, presentation, storage and transfer of all this information.

All these systems have firstly existed as purely centralized systems. Then, later when communication systems performances grew, communication capabilities have been added.

Currently, it is possible to remotely or locally access many multimedia data by using a high number of multimedia applications.

Thanks to the great deployment of the internet, new applications allow the interchange of information among users, or the access to distant multimedia information. These multimedia applications must take into account: on one hand, the users' requirements and the satisfaction degree that those applications can provide; on the other hand the services that the providers can offer and the requirements that they can effectively fulfill. In other words, these applications have to integrate the quality expected by the users for their multimedia services and the quality offered by the service providers.

1.1.2. Quality of service

Quality is defined by the ISO-9000 standard as "the degree to which a set of inherent characteristics fulfills requirements". Another ISO standard defines quality as "the totality of characteristics of an entity that bear on its ability to satisfy stated and implied needs" [ISO 8402].

Concerning the Quality of Service (QoS), ITU-T E.800 recommendation introduces a user/service approach and defines QoS as "the collective effect of service performance which determines the degree of satisfaction of the service user".

In the context of information technology and multimedia systems, QoS is defined as "the set of qualitative and quantitative attributes of a distributed multimedia system that is necessary to achieve the required functionality of an application" [VOG95]. These definitions drive us to distinguish two principal points of view: on one hand, the point of view of qualitative and quantitative attributes expected by the user and the one effectively perceived by the user; and on the other hand, the attributes of the QoS that a service provider attends to offer and the ones that are effectively provided.

1.1.3. Network solution

Traditional communication architectures typically provide a reliable data transport. Such a service is destined to applications having no time restrictions, for example still images, file transfers, a chat communications, among others; nowadays this kind of communication architectures are the most current. TCP/IP architecture is the best example. This architecture was able to satisfy all the traditional applications until the appearance of distributed applications having temporal constraints. The services required by these new applications have motivated the creation of new communication architectures. These new architectures aim at providing different services, essentially services able to guarantee not only the data delivery, but the delivery delay. Multimedia distributed applications requirements can only be satisfied by services fulfilling these two qualities: data delivering and delivering delay.

Unfortunately, the networks currently used for data transmission on the internet only accept bits sequences depending on the network capacity. Thus, the improvement of data delivery time can only be done by reducing data reliability. Indeed, reliability and delay are opposed attributes. It is therefore, from a given capacity, necessary to accept either a completely reliable,

but slow architecture, or either a fast but unreliable one. In fact, there exists an intermediate solution.

1.1.4. Partial Quality of Service

Depending on the type of media, the required satisfaction degree can change. For example, in a videoconference, a user can accept a low quality for the video flow, while the same quality degree for the audio flow would endanger the comprehension. It means that in a multimedia communication, some flows are more important than others, and that the QoS for less important flows can be reduced in order to improve the QoS of the first ones. It also means that in a multimedia communication, the QoS for each single flow is not always the same. It is so possible to talk about a Partial Quality of Service. This partial QoS can be defined in terms of per second accepted Application Data Units (ADU), for example.

In a real-time oriented, and so non reliable communication architecture, the partial QoS can be defined in terms of loss per second, total acceptable losses, etc. A model representing this partial quality of service is POC (see Figure 1) [DIA94].

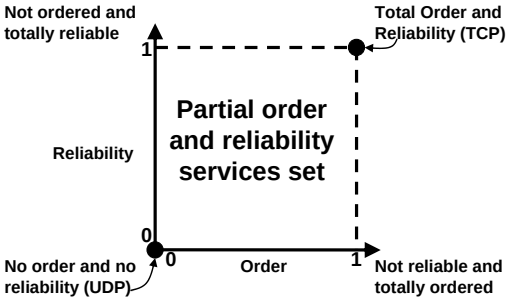

Figure 1: POC Model

This partial order and reliability model thus makes evident the necessity of new point-to-point protocols and services in order to guarantee the partial quality of service for the multimedia applications (to guarantee the flows with strong temporal constraints).

Once defined the problem of partial reliability on point-to-point multimedia data transmission, the idea of sending these same data to a set of users at the same time and with the same QoS constraints comes by itself; in other words, this leads to the need of a partial QoS multimedia multicast communication.

1.1.5. End-to-End Multimedia Point-to-Point Transmission

This section presents a survey of some transport and network communication services; this is done in order to show the necessity of new transport protocols, better adapted to time sensitive data flows characteristics.

The Open System Interconnection (OSI) reference model describes how information from an application in one computer moves through a communication medium to an application in another computer. This OSI model was developed by the International Standardization

Organization (ISO) in 1984 and it is now considered as the architectural reference model for interconnected computers. This model consists of seven layers, where the most important, layers 1 to 4, represent the very basic parts of communications systems. This section will describe some of the most used transport and network protocols.

1.1.5.1. *UDP/TCP*

The User Datagram Protocol (UDP) [POS80] is a minimal message-oriented transport layer protocol that is currently documented in IETF RFC 768. In the TCP/IP model, UDP provides a very simple interface between a network layer below and an application layer above. UDP provides no guarantees for message delivery and a UDP sender retains no state on UDP messages once sent onto the network. UDP adds only application multiplexing and data *checksumming* on top of an IP datagram.

Transmission Control Protocol (TCP) [POS81] is a connection-oriented, reliable delivery byte-stream transport layer communication protocol, currently documented in IETF RFC 793. It does the task of the transport layer in the simplified OSI model.

In the Internet protocol suite, TCP is the intermediate layer between the Internet Protocol below it, and an application above it. Applications most often need reliable pipe-like connections to each other, whereas the Internet Protocol does not provide such streams (but rather only unreliable packets).

Lacking reliability, UDP applications must generally be willing to accept some loss, errors or duplication. Most often, UDP applications do not require reliability mechanisms and may even be bothered by them. Streaming media, real-time multiplayer games and voice over IP (VoIP) are examples of applications that often use UDP. If an application requires a high degree of reliability, a protocol such as the TCP has to be used instead, or the application has to handle itself the needed reliability.

As UDP and TCP are not appropriate for many applications, newer transport layer protocols are being designed and deployed to address some of their inherent weaknesses. For example, real-time applications often do not need, and will suffer from TCP's reliable delivery mechanisms. In those types of applications it is often better to deal with some losses, errors or congestions than to try to avoid them.

1.1.5.2. *DCCP*

The Datagram Congestion Control Protocol (DCCP) [KOH02] is a message-oriented transport layer protocol that is currently under development in the IETF and it proposes an alternative to TCP and UDP. It offers an unreliable service (just as UDP) but within a connection and with a congestion control mechanism. This protocol, destined principally for interactive applications that are now using UDP, offers the possibility to choose the congestion control algorithm to be used.

The contributions of DCCP with respect to TCP/UDP are not limited to this possibility to choose the congestion control algorithm; it has been introduced other mechanisms such as the possibility to dynamically choose the acknowledgement frequency (ACK-Ratio), the acknowledgement vector (ACK-Vector), a low constrained flow control (Data Dropped), and the possibility to separately choose the congestion control algorithm for each transmission way.

1.1.5.3. *SCTP*

Stream Control Transmission Protocol (SCTP) [STE00] is an end-to-end, connection-oriented protocol that transports data in independent sequenced streams. SCTP endpoints support multi-homing; therefore, interface redundancy is built into the protocol. Through selective retransmission mechanisms, SCTP resolves errors and buffers losses in the data transmission process.

SCTP provides applications with enhanced performance, reliability, and control functions. This protocol is essential when detection of connection failure and associated monitoring is mandatory. Furthermore, SCTP could be implemented in network systems and applications that deliver voice/data and have to support quality real-time services (e.g., streaming video and multimedia).

The Signaling Transport (SIGTRAN) group of the Internet Engineering Task Force (IETF) defines SCTP standards in RFC 2960. The underlying mechanism of SCTP is fairly complex and incorporates a number of validation procedures, path-management practices, and security measures.

SCTP provides numerous advantages over UDP and TCP. For instance, SCTP combines the datagram orientation of UDP with the sequencing and reliability of TCP. Additionally, SCTP uses multi-stream, message-oriented routing in multi-homed environments.

1.1.5.4. *Intserv*

The IETF “Integrated Services” working group was created in 1994 in order to define an enhanced internet IP service model. This model was intended to transform the internet into an integrated services network, i.e. to transport efficiently audio, video, real-time data and classical data flows.

IntServ [BRA04] is a model for providing QoS on the Internet and intranets by using bandwidth reservation techniques. As originally designed, the Internet supports only best-effort delivery of data packets across multi-access (shared) network links. There is little support for QoS due to the packet-oriented nature of the Internet and factors such as variable queuing delays and congestion losses.

The IntServ model defines methods for identifying traffic flows, which are streams of packets going to the same destination. An Internet voice call is an example. The IntServ concept reserves just the right amount of bandwidth to support the flow's requirements and protect it from disruptions caused by network congestion. Reservations are negotiated with each network device along a route to a destination. If each device has resources to support the flow, a reserved path is set up. RSVP (Resource Reservation Protocol) is the signaling protocol that sends messages in the forward direction to request reservations, and then sends messages in the reverse direction to set up the reservations if all devices in a route agree to reserve resources.

IntServ is a bandwidth reservation technique that builds virtual circuits across the Internet. Bandwidth requests come from applications running in hosts. Once a bandwidth reservation is made, the bandwidth cannot be reassigned or preempted by another reservation or by other traffic. IntServ and RSVP are stateful, meaning that RSVP network nodes must coordinate with one another to set up an RSVP path, and then remember state information about the flow. This can be a very difficult task on the Internet, where millions of flows may exist across a router.

The RSVP approach is now considered too heavy for the Internet, but appropriate for smaller enterprise networks

1.1.5.5. *Diffserv*

The Differentiated Services (DiffServ) [BLA98] model for QoS was developed to differentiate IP traffic so that the traffic relative behavior could be determined on a per-hop basis.

By using DiffServ, traffic is classified based on priority. Then the traffic is forwarded using one of three IETF-defined per-hop behavior (PHB) mechanisms. This approach allows traffic with similar service characteristics to be passed with similar traffic guarantees across multiple networks, even if the multiple networks don't provide the same service the same way. This is an important feature because the Internet is really a network of multiple service provider networks.

DiffServ replaces the first bits in the ToS byte with a differentiated services code point (DSCP). The DSCP is then mapped to the PHB. This technique allows service providers to control how the DSCP codepoints are mapped to PHBs, and each time a packet enters a network domain it may be re-marked.

In spite of the progress obtained by these services, the complexity introduced in their deployment is so important that most of internet users have only access to traditional best-effort network services where the QoS constraints are not guaranteed.

1.1.6. Conclusions

So, as QoS network layer protocols do not exist, and that present transport protocols are too limited, it is possible to affirm that new transport protocols are needed in order to better provide QoS to multimedia applications. One of these new generation protocols is FPTP.

1.2. Fully Programmable Transport Protocol

1.2.1. Introduction

FPTP is a configurable and programmable protocol that provides communication services which satisfy the QoS constraints of distributed multimedia applications. The FPTP services are performed by the deployment of new transport mechanisms and by the configuration of those existent.

FPTP is a QoS oriented new generation transport protocol. It intends to provide a set of important transport mechanisms oriented to application requirements by using the existent network services and resources in an optimized way.

1.2.2. Design principles

FPTP has been designed by using the *Unified Development Process* and the UML and SDL languages. This methodology has permitted to define a contextual model of the QoS followed by a detailed services specification such as the description of the protocol structure and behavior. It has been tested and evaluated by simulations and real-scale experiments. It has also been proposed for its deployment a methodology based on programmable nodes [EXP03].

1.2.3. Quality of Service contextual model

FPTP has been specified within a standard QoS context, capable of providing a semantic space and an extensible architecture. This modeling principle makes possible the protocol composition, extension and specialization with the purpose of satisfying most of applicative needs, all this by using and extending, in an implicit way, the existent services. The contextual bases of this model are presented below.

1.2.3.1. *Quality of Service specification*

The QoS requirements and the corresponding actions when it cannot be ensured must be expressed within the QoS specification:

- By the per-flow QoS. A flow represents the data units composing a media (i.e. audio, video, data, etc). The flow QoS can be specified by the required bandwidth for its transmission, the delay, the acceptable loss rate and the admissible disorder between data units. It is also necessary to express the synchronization level among different flows within a session.

- By the QoS policy, i.e. the tolerance of a partially ordered or reliable service and the actions to be taken when they are not accomplished.

1.2.3.2. *Quality of Service Mechanisms*

The Quality of Service mechanisms can be classified into two categories: the static mechanisms related to the QoS provisioning; and the dynamic ones which control the QoS in the transfer phase. Table 1 shows this classification; mapping mechanisms are into the first category while control and management ones are in the second category.

Provisioning (static mechanisms)	
Mapping	QoS derivation or translation into transport parameters
Admission and deployment	QoS request acceptation from available resources evaluation. These mechanisms also perform the control and management of the mechanisms deployment in order to guarantee the required QoS
Control (dynamic mechanisms)	
Flow regulation	Mechanisms used for flow control based on the QoS specification (i.e. bandwidth, delay, reliability, order, synchronization, etc.).

Available resources control	Flow and congestion control: controls the data transmission rate based on the available resources and the receiver capacities
Management (dynamic mechanisms)	
QoS monitoring	Verifies the QoS offered by control mechanisms. The management mechanisms send feedback signals when a QoS parameter is out of the defined limits described in the specification. It is also possible to send signals to the control mechanism in order to indicate the QoS degradation.

Table 1: Quality of Service Mechanisms Classification

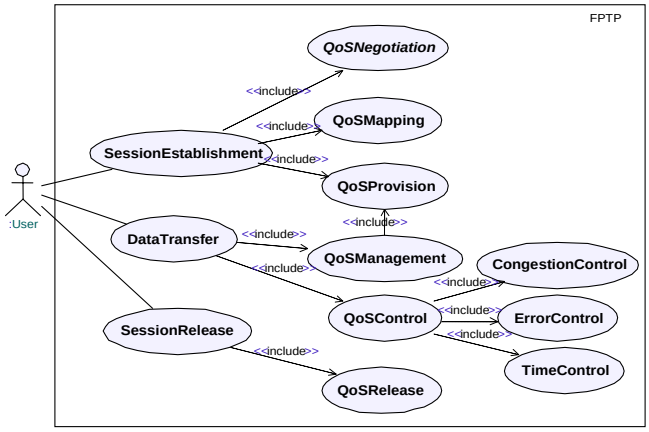

Figure 2: FPTP services UseCase diagram

Figure 2 shows the FPTP usecase diagram. *SessionEstablishment* includes the negotiation services, the requirements mapping and the transport mechanisms provisioning. In the *DataTransfert* phase, the control and management mechanisms are designed to guarantee in an optimum way the multimedia session QoS (i.e. congestion, error and temporal constraints control). The *SessionRelease* usecase takes in charge the session ending and resource freeing (a full explanation can be found in [EXP03]).

1.2.4. Protocol specification

The conception of a new programmable and extensible transport protocol must include a set of principles related to the applicative interface, the mechanisms and the protocol architecture.

- The FPTP Application Programming Interface (API) is defined as an extension of the BSD standard socket interface. This choice has been taken in order to keep the compatibility with the existent multimedia applications and, at the same time, enable the new multimedia applications to explicitly define their requirements in terms of QoS.

- The transport mechanisms deployed by this protocol must include some control mechanisms in order to preserve the network resources and some error control mechanisms in order to satisfy the applicative constraints. In order to deploy this protocol, some error control mechanisms have been selected to offer partially ordered (PO) and partially reliable (PR) services, together with the TFRC (TCP-Friendly Rate Control) congestion control mechanism. The error and congestion control

mechanisms have been extended in order to take into account the intrinsic multimedia flow characteristics and the application time constraints. In order to satisfy the timed application requirements in terms of adaptive multimedia flow reliability, it has been proposed a differentiated and partially reliable service. This service has been specialized to provide a partially reliable and differentiated service which takes into account the specified time constraints with the goal of correctly satisfying the specific application requirements constraints. In the same way, TFRC has been extended by replacing its usual delaying strategy by a new one based on a timed selective deletion. This extension provides a differentiated time constrained congestion control mechanism. The services resulting from error and congestion control mechanisms combination offers a huge transport services range. Figure 3 summarizes the FPTP mechanisms from the point of view of applicative requirements and network constraints.

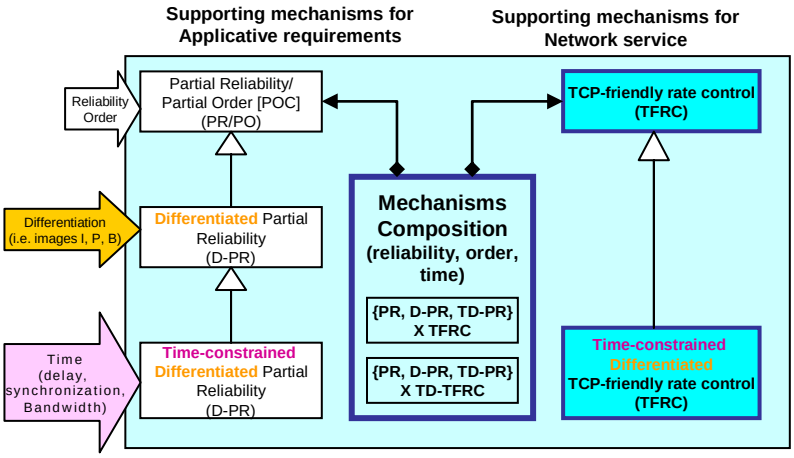

Figure 3: FPTP mechanisms set from the point of view of applicative requirements and network constraints

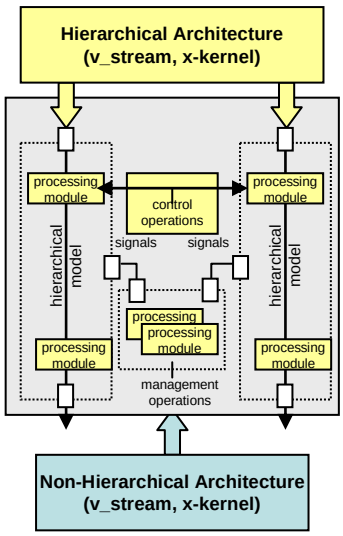

Figure 4: FPTP architecture

- FPTP protocol has to be deployed on a composable architecture able to combine and deploy the needed control and management mechanisms with the goal of providing a set of different transport services. FPTP architecture follows a hierarchical architecture for the control mechanisms and a non hierarchical one for the management mechanisms (see Figure 4).

1.3. On the Multicast

1.3.1. IP Multicast

Without a doubt, group communication (the one-to-many or many-to-many delivery of data), also called Multicast, is the focus of an intense study in the internet research community. The proliferation of multimedia applications associated with new high-speed networks is raising the need of group communication mechanisms and protocols.

Systems created in the early internet were designed to support point-to-point services. A point-to-point communication implies two participants, one of which is typically a server and the other is a client. In these systems, group communications were mostly restricted to LAN environments and were well supported by existent local area network technologies, such as Ethernet and Token Rings. Nonetheless, interconnections among extended LANs were performed by bridges with no support for multicast. Although multicast addressing was as a separate address class from the beginning, there were no standard ways to use it.

In the late 80s, Deering proposed multicast extensions to the unicast routing mechanisms across datagram-based inter-networks [DEE90], marking the beginning of IP multicast.

Deering's work led to the creation of the Multicast Backbone (MBone) [ERI94]. In March 1992, the MBone carried its first event with 20 sites worldwide receiving multicast audio streams from a meeting of the Internet Engineering Task Force (IETF) in San Diego [ALM00].

The MBone is a set of workstations running a daemon called *mrouted*, which receives unicast-encapsulated multicast packets on an incoming interface and then forwards packets over the appropriate set of outgoing interfaces. Connectivity among workstations was provided using point-to-point IP-encapsulated tunnels. Each tunnel connected two endpoints via one logical link, but could cross several Internet routers. Once a packet is received, it can be sent to other tunnel endpoints or broadcast to local members. Routing decisions were made using the Distance Vector Multicast Routing Protocol (DVMRP) [WAI88].

The original multicast routing protocol, DVMRP, creates multicast using a technique known as *broadcast-and-prune* or *flood-and-prune* which makes this protocol unscalable.

In DVMRP, each router discovers the existence of group members by periodically issuing an Internet Group Management Protocol (IGMP) query. Upon receiving the query, a leaf router will send a prune message indicating that it does not have directly attached group members. An intermediate router forwards the prune message towards the source if it receives prune messages on all its interfaces except the interface towards the source. Such a mechanism requires that every router that supports multicast, has to keep a state for each existing multicast group, regardless if the router itself actually belongs to the group tree or not. Thus DVMRP is also

referred to as the *dense mode* protocol, as it assumes the dense spreading of group members where pruning is rare [SHE02].

Since 1992, the MBone has grown and evolved a lot. The appearance of native multicast, i.e. routers capable of handling multicast packets, has revealed the inefficiency of dense mode. Two additional dense mode protocols were developed as a result of ongoing research: Multicast Extensions to Open Shortest Path (MOSPF) [MOY94] and Protocol Independent Multicast – Dense Mode (PIM-DM) [DEE96]. The explanation of one of them can be extended in order to understand the other. PIM-DM is very similar to DVMRP; there are only two major differences. The first is that, while DVMRP maintains it own routing table, PIM-DM uses whatever unicast table is available. The name PIM is derived from the fact that the unicast table can be built using any unicast routing algorithm. The second difference is that DVMRP tries to avoid sending unnecessary packets to neighbors who will then generate prune messages based on a Failed Reverse Path Forward (RPF) check. The set of outgoing interfaces built by a given DVMRP router will include only those downstream routers that use the given router to reach the source. PIM-DM avoids this complexity, but the trade-off is that packets are forwarded on all outgoing interfaces [ALM00]. Thus, the principal problems with dense mode are: unnecessary messages sent to the network and large multicast tables, even on those routers which do not belong to the session.

These disadvantages motivated a new class of multicast routing protocols: the Sparse Mode multicast routing protocols. Instead of optimizing only for the case when a group has many members, sparse mode protocols are designated to work more efficiently when there are only a few widely distributed group members. In those protocols, receivers are expected to send explicit join messages and data traffic to a router acting as a core. The use of a core as a "meeting place" for sources and receivers facilitates the creation of the multicast tree. Two of the most popular sparse mode protocols are the Core Based Trees (CBT) [BAL95] and Protocol Independent Multicast – Sparse Mode (PIM-SM) [EST98], the later being the most widely implemented. Sparse mode protocols have a number of advantages over dense mode protocols. First, sparse mode protocols typically offer better scalability in terms of routing state. Only routers on the path between source and a group member must keep states (dense mode protocols require states in all routers in the network). Second, sparse mode protocols are more efficient because the use of explicit join messages means that multicast traffic only flows across links that have been explicitly added to the tree [ALM00].

In spite of numerous efforts of a generation of researchers, there remain many unsolved issues in the IP multicast model that delay the development and deployment of the IP multicast protocol and of multicast applications. Furthermore, the diversification of new applications, the growth of multicast users and the appearing of new very different internet-capable devices (PDAs, cellular phones and other mobile hosts, for example) have led new requirements and constraints for multicast protocols and mechanisms. The most prominent issues are the lack of a multicast address allocation scheme, the lack of a membership management and access control, and the lack of ordering, synchronization, security and QoS mechanisms.

First, the class D portion of the IP addresses space (a 32-bit number in the range of 224.0.0.0 and 239.255.255.255) has no geographical or topological meaning. This flat addressing schema is non-scalable and makes the packet routing very hard.

Second, in present internet there is no hierarchy in the group structure. A group address is chosen randomly and it is used hopping it is not currently in use. The possibility of addressing conflicts increases with the number of multicast groups and complicates the application

unnecessarily. In present multicast, any host can send to a multicast address without registering itself within the group.

To reduce these complexities, a new generation of multicast protocols emerged to support a subclass of multicast applications: single source multicast applications. By restricting to a single source multicast, a multicast group, which is also called a *channel*, is indicated by a pair of source and group addresses. This allows sources to select a locally unique group address which, together with the source's own IP address, will uniquely identify the multicast channel. Thus, SSM solves some of the above mentioned issues such as the address allocation problems and the control of the data sources. This single source model argues that at least in the near future, large scale Internet broadcast service will dominate the multicast service market.

Third, these lacks on registration management led to security problems since any node can send/listen to/from any multicast address without registering. Some sender control is necessary in order to avoid possible attacks. Then, some receiver restrictions mechanisms are also necessary to provide confidentiality.

Fourth, IP Multicast is based on the well known best-effort schema; then, no ordering, synchronization or reliability mechanisms are provided. For many distributed applications, an ordered reception of packets is required, because any disordering may give a different view of the state of the group. A packet reordering mechanism is then necessary. Nevertheless, this ordering requirement can be relaxed for some multimedia applications [AME94].

Reliability is another important point for some applications. Usually, reliability implies packet recuperation and ,as a consequence, delay augmentation. In point-to-point applications it is possible to find a compromise among these two features, but in group communications this compromise is not easily found. It is then necessary to define a hierarchy for the hosts and to apply a partial reliability mechanism. Finally, QoS and multicast-service deployment are unsolved tasks in current multicast.

As these two problems, i.e. group management with partial reliability and protocol deployment are to be solved in this work, they are being introduced in next subsections and completely deployed in next chapters.

1.3.2. IP Multicast Deployment: An Overlay Tree Solution

The conventional wisdom has been that IP is the natural protocol layer for implementing multicast related functionality. However, a set of factors have limited the ubiquity of IP multicast services.

First, as said before, more than a decade after its initial proposal, IP Multicast is still plagued with concerns pertaining to scalability, network management, deployment and support for higher layer functionality such as error, flow and congestion control [CHU00].

Second, in the internet architecture, the internetworking layer, or IP, implements a minimal functionality, a best-effort unicast datagram service, and end systems implement all other important functionality such as error, congestion and flow control. IP multicast is the first significant feature that has been added to the internetworking layer since its original design and most routers today implement IP Multicast. However, providing higher level features such as

reliability, congestion control, flow control, Quality of Service and security has been shown to be more difficult than in the unicast case.

Third, IP Multicast calls for changes at the infrastructural level, and this slows down the phase of deployment [CHU00]. Among the business-oriented factors, there exists the fact that there is no standard method to charge for multicast services. Which parameters to base the fees on? Several parameters are possible candidates, such as the group membership, or the number of packets exchanged, or the bandwidth consumed, yet as such no scheme has been proposed to standardize the calculation of the fees for multicast service.

All these reasons, along the fact that multicast-enabling might often require routers to forward data that is not required in their own local Autonomous System (AS), are the major reasons why ISPs are reluctant to install multicast-enable routers, or even configure multicast-capable routers to provide this service.

When the network infrastructure does not support the IP Multicast routing protocols, it is possible to use a new technique called Application Level Multicast (ALM) [PEN01] which is based on Overlay Networks (ON). In the broadest sense, an *Overlay Network* is a set of "tunnels" formed among network edges to support a common packet processing function other than the ones supported in the conventional network. These tunnels are unicast connections setup among the service nodes on top of the general network infrastructure. The primary advantage of the overlay network architecture is that it does not require universal network support (which has become increasing hard to achieve) to be useful. This enables faster deployment of desired network functions and adds flexibility to the service infrastructure, as it allows the co-existence of multiple overlay networks, each supporting a different set of service functions. ALM is also known as an application-based distribution and is used to provide multicast distribution services in situations where no support is provided by the network for multicast.

By using ALM to implement multicast communication, each participating end system is responsible to forward received datagrams to other ALM members in the multicast group.

One of the advantages of using this technique is the possibility of dynamically configuring the end system to either retransmit or filter every single datagram it receives. This model is used by some multicast protocols such as TBCP [MAT01] or [CHU00, WRI00, LEE01].

Nonetheless, none of these works take into account the user QoS in order to calculate the multicast tree structure.

We can remark that most of LANs are native multicast capable, while Internet (the core network) is not. So, it makes sense to try to relay all these native multicast LANs by a set of unicast tunnels (i.e. an Overlay Network) forming a tree by following the MBone model. For doing so, it would be necessary to define an access point, a sort of proxy on each LAN, and this access point would be within the ON. Next step is to forward data from one sender proxy to all other access points. It has been explained that a flooding solution is not adequate; so it would be necessary to create a spanning tree on the top of the ON in order to economize resources. A network model like this is shown in Figure 5.

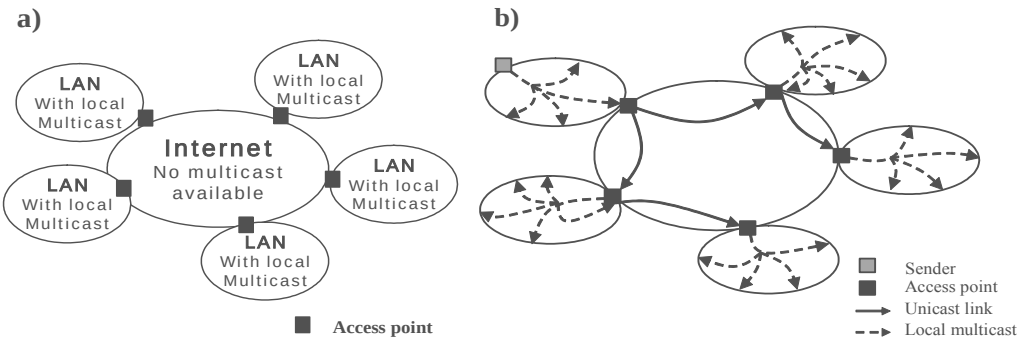

Figure 5: Network Model

Next subsection gives a brief survey of different works targeting the spanning tree creation.

1.3.3. Spanning Trees Survey

Several fundamental problems in the design of communication networks can be modeled as finding a network obeying certain connectivity specifications. For instance, the network may be required to connect all the nodes in the graph (a spanning tree problem), a subset of the nodes in the graph (a Steiner tree problem [VOS92, WIN92, SMI92]) or may be only interconnect a set of site-pairs of nodes (a generalized Steiner forest problem) [RAV93]. The goal in such network-design problems can usually be expressed as minimizing some notion of "cost" associated with the network or some constraints imposed by the applications or the users (for instance, the QoS).

There are three classic examples of such cost measures among others. If we associate costs with feasible edges and nodes that can be used to build the network, then we may look for a network such that the price of construction is minimized. This is the *minimum-cost design* problem or Minimum Spanning Tree (MST) in the graph theory. There exist many well known algorithms for finding such a tree, e.g. Kruskal's [KRU56] or Prim's [PRI57] algorithms. A notion of cost that reflects the vulnerability of the network to single point failures and also quantifies the amount of decentralization in the network is the maximum degree of any node in the network. Minimizing this cost corresponds to the *minimum-degree network design* problem. A notion of cost that reflects the end-to-end delay and deals with minimizing this end-to-end delay from a single node, called source, to the rest of nodes in the network, is called Single Source Shortest Path Tree (SPT). A single end-to-end delay is the sum of every single link delay in the path from node source to a vertex in the graph. Dijkstra's [DIJ59] algorithm can solve the SPT problem.

Finding a network of sufficient generality and of minimum cost with respect to each one of these measurements can be shown to be NP-complete. Hence, much of the work focuses on approximation algorithms for each of these problems. However, in applications that arise in real-world situations, it is often the case that the network to be built is required to minimize more than one cost criterion at once.

In the case of mono-criterion algorithms, the most used are the Greedy algorithms. An algorithm that always takes the best immediate or local solution while finding an answer is said to be “Greedy”. These algorithms find the overall, or globally, optimal solution for some optimization problems, but may find less-than-optimal solutions for some instances of other problems.

Greedy algorithms work in phases. In each phase, a decision is made that appears to be good, without regard for future consequences. Generally, this means that some *local optimum* is chosen. This “take what you can get now” strategy is the source of the name for this class of algorithms. When the algorithm terminates, we hope that the local optimum is equal to the *global optimum*. If this is the case, then the algorithm is correct; otherwise, the algorithm has produced a sub-optimal solution. If the best answer is not required, then simple greedy algorithms are sometimes used to generate approximate answers, rather than using the more complicated algorithms generally required for generating an exact answer. Prim's algorithm [PRI57] and Kruskal's algorithm [KRU56] are greedy algorithms that find the globally optimal solution for the *minimum spanning tree* while Dijkstra’s algorithm [DIJ59] does it for the *shortest paths three*.

Let us see some examples of how these algorithms work. A network model is given in Figure 6a and, for all algorithms, the vertices are added in the given order (V1, V2, …, V13).

Kruskal's algorithm is an algorithm in graph theory that finds a minimum spanning tree for a connected weighted graph. This means that it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized. If the graph is not connected, then it finds a *minimum spanning forest* (a minimum spanning tree for each connected component).

It works as follows:

- To create a forest *F* (a set of trees), where each vertex in the graph is a separate tree
- To create a set *S* containing all the edges in the graph
- while *S* is nonempty
    - To remove an edge with minimum weight from *S*
    - if that edge connects two different trees, then add it to the forest, combining two trees into a single tree
    - otherwise discard that edge

At the termination of the algorithm, the forest has only one component and forms a minimum spanning tree of the graph.

With the use of a suitable data structure, Kruskal's algorithm can be shown to run in O (m log m) time, where m is the number of edges in the graph. An example is provided in Figure 6b.

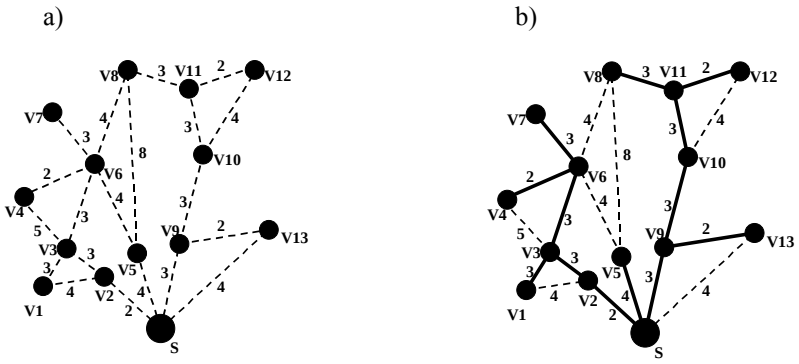

Figure 6: a) Example model. b) Minimal Spanning Tree.

Prim's algorithm is an algorithm that also finds a minimum spanning tree for a connected weighted graph. If the graph is not connected, then it will only find a minimum spanning tree for one of the connected components.

It works as follows:

- create a tree containing a single vertex, chosen arbitrarily from the graph
- create a set containing all the edges in the graph
- loop until every edge in the set connects two vertices in the tree
  - remove from the set an edge with minimum weight that connects a vertex in the tree with a vertex not in the tree
  - add that edge to the tree.

Prim's algorithm can be shown to run in time which is O (m + n log n) where m is the number of edges and n is the number of vertices. An example is provided in Figure 6b (let us recall that this algorithm, such as Kruskal's one, obtain an SPT and so the obtained tree is the same).

Naive algorithms are those which follow the problem solving meta-heuristic of making the locally optimum choice at each stage with the hope of finding the global optimum. For instance, applying the naive strategy to the traveling salesman problem yields the following algorithm: "At each stage visit the nearest unvisited city to the current city".

Naïve algorithms are Greedy and they rarely find the globally optimal solution consistently, since they usually don't operate exhaustively on all the data. Nevertheless they are useful because they are quick to run and often give good approximations to the optimum. An example of a simple greedy algorithm can be seen in Figure 7a.

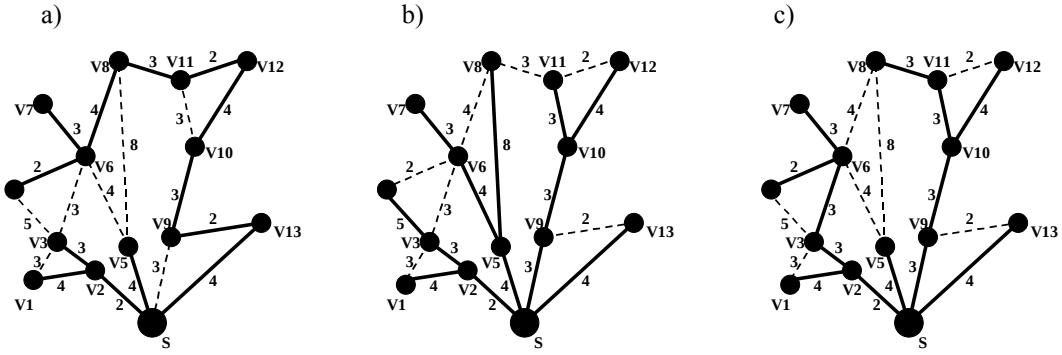

Figure 7: a) Naive Tree; b)SPT-Dijkstra's Tree; c) SBPT Tree

Dijkstra's algorithm, Solves a shortest path problem for a directed and connected graph G(V, E) which has nonnegative edge weights.

The algorithm works by constructing a subgraph *S* such that the distance of any vertex *v'* (in *S*) from a source vertex *s* is known to be a minimum within *G*. Initially *S* is simply the single vertex *s*, and the distance of *s* from itself is known to be zero. Edges are added to *S* at each stage by (a) identifying all the edges $e_i = (v_{i1}, v_{i2})$ in *G-S* such that v_{i1} is in *S* and v_{i2} is in *G*, and then (b) choosing the edge $e_j = (v_{j1}, v_{j2})$ in *G-S* which gives the minimum distance of its vertex v_{j2} (in *G*) from *s* from all edges e_i. The algorithm terminates either when *S* becomes a spanning tree of *G*, or when all the vertices of interest are within *S*. An example of this kind of tree can be seen in Figure 7b.

Shortest Best Path Tree (SBPT). This algorithm makes a tradeoff between a simple naive algorithm and a SPT. It connects the nodes by using the shortest path (as the SPT), but when there are more than one possible paths with the same length, then it chooses the closer neighbor. An example of this kind of tree can be seen in Figure 7c.

On the other hand, many works have dealt with optimizing more than one cost criterion simultaneously. In [WIN92, BAU95, KRU96], it has been treated the problem of Steiner multicast trees while [RAM96] deals with the problem of multicast trees on networks with asymmetric links.

Ravi et al [RAV93] introduce several problems in network design involving multiple objectives and provide approximations for many of them; particularly two cost measures have been minimized simultaneously: the total cost of the network and the maximum degree of any node in the network. The authors take into account an undirected graph and the goal is to span a subset of the entire graph. The goal for minimizing the maximum degree of any node in the network is the reduction of the depth, which reduces the number of hops in the communication. Their techniques extend to the Steiner tree and generalized Steiner tree problems with the same ratio. They also presented efficient algorithms for computing subgraphs that have low weight and small bottleneck cost.

Radha et al [RAD01] propose an algorithm to find a Directed Minimum-Degree Spanning Tree with a quasi-polynomial time approximation. The authors present an approximation algorithm for the directed minimum-degree spanning tree problem and introduce notions of witness sets that work in directed graphs. An important remark on their work is that their graph is a non-weighted one.

Ito et al [ITO02] define and propose a solution for the *file transfer tree* (FTT) problem. FTT consists in finding a shortest path T rooted at r such that for each vertex in T, the number of children does not exceed the capacity $\delta(v)$ in a directed, weighted, acyclic graph. This paper gives a good study of some FTT extensions by relaxing the vertex degree or the distance to the source constraints in various ways and show polynomial algorithms for the computational hardness of these problems.

Gang et al [GAN99] present two heuristics for constructing delay constrained multicast trees in directed networks. The first one called "Delay-constrained Shortest Path Multicasting" is a static minimum cost tree computation algorithm that satisfies the end-to-end delay requirements set by the application. The second one called "Dynamic Delay-constrained Multicasting" is an efficient dynamic multicast routing algorithm that can handle multicasting dynamics such as the joining of nodes during an existing session. This algorithm overcomes the others on what it manages dynamic groups. Nevertheless, it is oriented to single criterion problems.

Mathy et al present an approach called Tree Building Control Protocol (TBCP) [MAT01]. TBCP is a generic protocol designed to build overlay spanning trees among participants of a multicast session, without any specific help from the network routers. TBCP therefore falls into the general category of ALM protocols and mechanisms. TBCP is an efficient, distributed protocol that operates with partial knowledge of the group membership and restricted network topology information. One of the major strategies in TBCP is to reduce convergence time by building as good a tree as possible early on, given the restricted membership/topology information available at the different nodes of the tree.

As said before, management of dynamic groups is important. Frigioni et al [FRI00] propose fully dynamic algorithms for maintaining distances and the shortest paths from a single source in either directed or undirected graph with positive real edge weights, handling insertions, deletions, and weight updates of edges. Authors offer also a study of single source shortest paths on general directed or undirected graphs.

After this brief survey of multicast methods and algorithms, next subsection explains how this work pretends to add user QoS to network models and multicast trees.

1.3.4. Including User QoS: A Hierarchized Graph for heterogeneous users

Several proposals [LOR02] [BIA03] [YAN02] have appeared in the literature in the area of QoS multicast. For example, In NARADA [CHU00], Chu et al state the disadvantages of implementing multicast functionality at the IP-level and propose moving the functionality to the application (end-system) level, where multicast is built on top of unicast using an overlay network. This addresses the problem of routers multicast state and provides a solution for reliability and congestion/flow control on top of unicast as it is a well-understood area, and allows application-specific solutions to be built-in.

For the authors, the drawbacks of end-system multicast are: *a)* Performance cannot be as good as IP multicast as redundancy cannot be completely eliminated, and delays are higher. *b)* Topological information must be extracted by end-systems to improve efficiency.

The paper then mentions two methods of implementing end-system multicast; firstly Peer-to-peer based architectures, and secondly Proxy-based architectures.

They compare the performance of their protocol with IP-multicast. Narada has been developed for being: Self-organizing, Overlay efficiency, Self-improving, Adaptive.

The parameters they use to measure performance under different circumstances include latency, bandwidth, stress, resource usage and protocol overhead. Narada attempts to build an efficient overlay structure by first building a good mesh, and then constructing per-source shortest path trees out of this mesh.

They conclude that end-system multicast is a viable option for small-medium sized groups on the internet. This work approximates to our objective, but they do not take the user QoS into account in order to construct the mesh or the per-source shortest path tree out of this mesh. Nonetheless, the idea of creating a per-source shortest path tree will be taken for our work.

Another excellent work dealing with QoS multicast is *Scattercast*. Here, Chalmers et al [CHA00] proposed an architecture for Internet broadcast distribution as an infrastructure service. In Scattercast, strategically placed service agents dynamically organized into a source-based distribution tree. Since most Content Distribution Networks (CDNs) are single-source applications, a source-based multicast tree fits them well. Another focus of Scattercast is the application transport layer that provides reliability and adapts to meet individual application constraints. These strategically placed service agents will also be taken within our approach, this will add different QoS for each user.

1.3.5. Conclusions

After this brief survey of multicast algorithms and some recent works on QoS multicast it is possible to remark that the QoS requested by the users is not considered as a parameter for deciding the multicast tree shape. This dissertation will propose in chapter 2 a network organization model as a graph which takes into account at the same time the users' QoS requirements and the network performances: this model is called Hierarchized Graph (HG). Then, in chapter 3 it will be proposed an algorithm based on Dijkstra's one which creates an SPT where the maximal output degree for each vertex is limited: this algorithm is called DgB-SPT.

Even with a mechanism allowing to solve the lack on IP multicast deployment (ALM), with a model representing the users' QoS constraints and the network performances (HG) and with an algorithm which creates a Degree-Bounded Shortest-Path-Tree (DgB-SPT), it reminds the problem of *effectively* deploying these solutions on access points. Next section describes a possible solution for this problem.

1.4. Dynamic deployment solution: Programmable networks

As said before, Multimedia group communication is increasing enormously, and multicast is widely recognized as an important service that enables efficient group communication. Nonetheless, multicast protocols are still not widely available, and the experimental Multicast Backbone (MBone) is slow to take off. Besides that, some users have no MBone access at all, providers are reluctant to allow IP Multicast in their networks, and firewalls can block multicast traffic [YAM01].

Motivated by new technologies and middleware paradigms in telecommunication networks, and the pressing need to accelerate network innovation, network researches are exploring new ways for network routers, switches, and base stations to be dynamically programmed by network applications, users, operators, and third parties. Network elements such as controlled-switches have been made programmable for some time but not in the dynamic manner requested by the current needs. Active and programmable networks seek to exploit advanced software techniques and technologies in order to make network infrastructure more flexible, thereby allowing users and service providers to customize network elements to meet their own specific needs. Customizing routing, signaling, resource allocation and accelerating information processing in this manner raises a number of significant security, reliability and performance issues.

Competition among existing and future Internet service providers (ISPs) may hinge on the speed at which one service provider can respond to new market demands over others. The introduction of new services is a challenging task requiring new tools for service creation, including new network programming platforms and supporting technologies. Future programmable networks are likely to be based on active and programmable networking and open signaling techniques. Both of these proposals squarely address the same problem: how to "open up" the network and accelerate its programmability in a controlled and secure manner for the deployment of new architectures, services, and protocols.

The separation of communication hardware (switching fabrics, forwarding engines) from control software is fundamental to making the network more programmable. Such a separation is difficult to realize today. Typically, service providers do not have access to switch/router control environments (e.g., the router's operating system), algorithms (e.g., routing protocols), or states (e.g., flow states). This makes the deployment of new network services, which may be many orders of magnitude more flexible than proprietary control systems, impossible due to the closed nature of network nodes [CAL01].

The work on open signaling (OPENSIG) exemplifies the development of new network programming environments that explicitly recognize service creation, deployment and management in the network infrastructure. Here, there is a clear distinction between the transport and control, and the objective is to make the control plane fully programmable. The work on active networks grows from the idea of allowing the traffic itself to program the network's behavior—an idea that is potentially applicable in various dimensions of programmability, for example in-band versus out-of-band or per-packet, per-flow, or per-network granularity. In its most general form, executable programs are embedded within data packets (called "capsules") and executed by network elements as they traverse the network. Programmable networks have evoked substantial interest in the networking community, standards bodies and industry. Standards initiatives include the IEEE 1520 programmable Interfaces for Networks, Multiservice Switching Forum, and IETF GSMP and FORCES initiatives on application programming interfaces (APIs) for routers. A growing number of international forums have arisen for presentation of advances in active and programmable networks, including OPENSIG, IEEE Conference on Open Architectures and Network Programming (OPENARCH), and IFIP International Working Conference on Active Networks (IWAN) [CAL01].

A programmable network is distinguished from other networking environments by the fact that it can be programmed from a minimal set of APIs to provide a wide array of higher level services.

A number of programmable network prototypes have been targeted to specific networking technologies. The motivation behind these projects is to make the targeted networking technology more programmable in an attempt to overcome particular deficiencies associated with supporting communication services. A number of research groups are actively designing and developing programmable network prototypes, for example: Smart Packets [KUL98], xbind [CHA96], Mobiware [ANG98], ANTS [WET98], Switchware [ALE98] and Netscript [ADA97], among others.

In the context of the European GCAP [GCAP] and the French @IRS++ [@IRS++] projects, some experiments on dynamic deployment have been performed. These tests exploited two programmable and active platforms: Simple Active Router-assistant Architecture (SARA) [SARA] and JavaProxy Active platform developed by 6wind [6WIND, DIA01, DIA02, DIA02-2, GAR02].

1.4.1. Simple Active Router -assistant Architecture

SARA implements a Node Operating System (NodeOS) and an Execution Environment (EE) over a dedicated processor, called Assistant, linked to an enhanced router. This approach permits to add active node functionalities to existing off-the-shelves routers in a safe and high-performance way [URU02].

The SARA platform can operate over any router in the middle of the network without the explicit knowledge of the user. This means that users send their active packets to the final destination and the routers recognize and process them following the corresponding code, all this while taking normal routing decisions for non active packets.

1.4.1.1. *SARA active node architecture*

In SARA, the active applications are not processed by the router, but by a dedicated processor called Assistant, linked to the router. This approach reduces the overhead in the router to the simple identification and redirection of the active packets. SARA is an active node prototype developed using JAVA and is able to transparently process the active packets passing through the router.

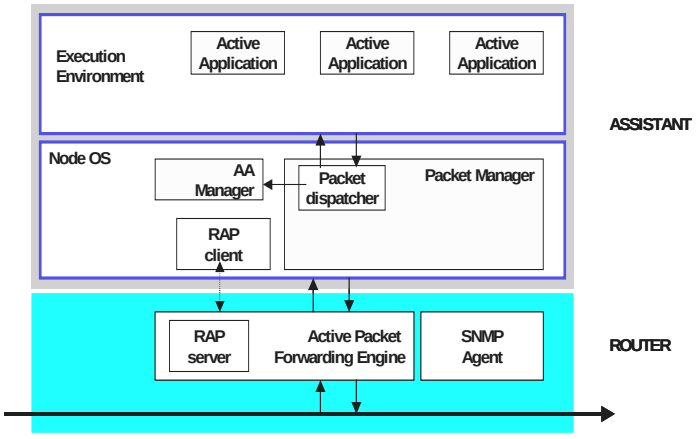

Figure 8: SARA Architecture

In agreement with the active general architecture described above, the SARA platform is formed by two principal modules, the Execution Environment (the execution support for the active applications), and the NodeOS. This NodeOS controls the entire system, administrates the applications and distributes the packets coming from the routing machine to the concerning applications (Figure 8).

1.4.1.2. *SARA Transparency*

Transparency of active node implementation is a desirable characteristic for allowing active services to be easily developed without disrupting distributed applications development practices.

The active packet concept in SARA is transparently implemented using the "router alert" bit within the IP header. The active packets are simply UDP datagrams encapsulated into IP packets with the router alert bit activated. The SARA header is shown in Figure 9.

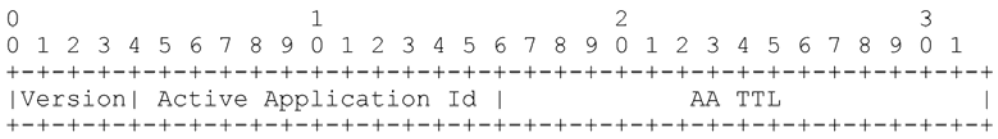

Figure 9: SARA packets

SARA provides packet encapsulation for requesting an active service by an UDP socket available in the SARA API.

1.4.2. JavaProxy Active Platform

JavaProxy is the Active Execution Environment implemented on SixOS Release that offers Java support on 6WINDGate [6WIND]. JavaProxy is basically a set of Java classes organized within different Java packages. It is an open framework that manages objects called "Modules". Modules are objects implementing custom services and are intended to be written by 6WINDGate users, namely Service Provider.

1.4.2.1. *Components*

JavaProxy actually is a framework for objects called "modules". Its main goals are:

- To dynamically download modules into the framework (and instantiate them).
- To particularly provide network services to these modules, namely full socket services:
  - Standard UDP / TCP socket services
  - IPv4 / IPv6 socket services
  - IPv4 IPv6 Divert socket services: this type of socket permits to develop transparent proxies and breaks the end-to-end model.
- To also provide full access to 6WINDGate's functions: QoS, Security, etc…

The following figure shows the Proxy Wrapper architecture:

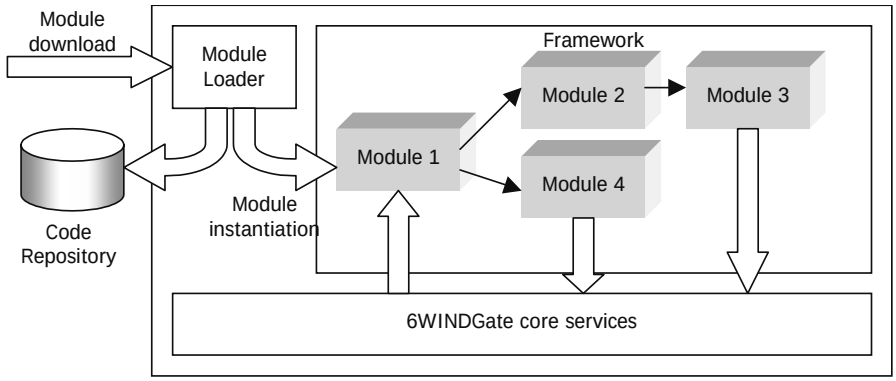

Figure 10: JavaProxy Architecture

As it is shown this architecture, JavaProxy is made of three components:

- The 6WINDGate Core Services
- The Active Module Loader
- The Module Framework.

1.4.2.2. *6WINDGate Core Services*

This component provides access to all functions of the 6WINDGate. Particularly, this service access focuses on network services, i.e. the socket services. But it also offers access the other 6WINDGate services: QoS, Security, etc…

Another important point is the access to the divert service that permit to divert classified packets from the kernel to the user space. This allows easily applying a custom process on the desired packets with no kernel customization.

1.4.2.3. *Active Loader*

This component performs the following tasks:

- It actively downloads needed code from remote servers
- It manages a local code repository to optimize useless code download
- It instantiates the needed Modules into the framework.

1.4.2.4. *Modules*

JavaProxy is actually simple and is based on the concept of Module. A Module is an independent and autonomous object that processes ADU (Application Data Unit).

Modules can be seen as “black boxes” with inputs and outputs that wrap user defined datagram processing methods. A datagram on input is processed and the resulting datagram (that can be uncorrelated with input datagram) is sent on outputs.

The behavior of the module defines the number of inputs and outputs, how inputs are read, how datagrams are processed and where they are forwarded to. A module may have multiple inputs and read them one by one or wait and read them synchronously; the module can forward datagrams to one or several outputs; the number of inputs and outputs may vary during the module lifetime.

All this theory is the base of the goal of this dissertation. It reminds to put all together into a single block. For this it is necessary to use some formal (or semi-formal) methods and techniques in order to correctly model, validate, verify and simulate. Next section gives an introduction to this problem and proposes the use of a software engineering method called MDA and a corresponding modeling language UML.

1.5. System modeling, validation and simulation

In the transition from the industrial age to the information age, software is indispensable. It is undeniable that systems requirements are growing incredibly; this fact joined with the migration of hardware tasks into software systems are making the traditional systems more and more complex. It is so necessary to increment, over all, the quality of produced software. The principal problem is to increment the software quality and reduce the required time to create it. One possible solution is the utilization of models, which are easier to test, modify and validate than real systems. An iterative and incremental software development process where the model of a system is iteratively refined into executable systems via a series of systematic mapping transformations is called *model driven development* [KOB03].

One of the main purposes of the model is to close the gap between different actors in the development process: requirements engineers, system analyst, software developers, and testers all speak the same language.

Modern tools for model driven development provide the capability of executing (parts of) a model, and this makes possible to get an early verification that the system works as intended. Testing also becomes a more important activity in model driven development, since it is applied much earlier and more frequently. This way, it is possible to verify that the different parts of an application will fit together at the end of the project [BJO02].

It is so impossible to deny the importance and the usefulness of models within system design, analysis, implementation, validation, maintaining and deployment.

1.5.1. Modeling Methodology

The Object Management Group (OMG), the world's largest software consortium, is promoting the model driven development through its Model Driven Architecture (MDA) initiative [MDA]. The OMG defines MDA as:

An approach to Information and Telecommunication (IT) systems specification that separates the specification of functionality from the specification of the implementation of the functionality on a specific technology platform

Figure 11 lays out the Model Driven Architecture (MDA), which is language-, vendor- and middleware-neutral.

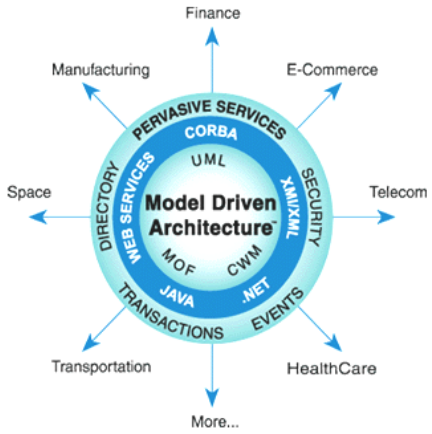

Figure 11: Model Driven Architecture

The core of the architecture, at the center of the figure, is based on OMG's modeling standards: UML, the MOF and CWM. There will be multiple *core models*: One will represent Enterprise Computing with its component structure and transactional interaction; another will represent Real-Time computing with its special needs for resource control; more will be added to represent other specialized environments but the total number will be small. Each core model will be independent of any middleware platform. The number of core models stays small because each core model represents the common features of *all* of the platforms in its category.

The first step when constructing an MDA-based application is to create a platform-independent application model expressed via UML in terms of the appropriate core model. Platform specialists will convert this general application model into one targeted to a specific platform such. Standard mappings will allow tools to automate some of the conversion. These target platforms occupy the thin ring surrounding the core.

Maximizing automation of the mapping step is a goal; however, in most cases some hand coding will be required, especially in the absence of MDA tools. As users and tool builders gain experience, and techniques for modeling application semantics become better developed, the amount of intervention required will decrease.

The platform-specific model faithfully represents both the business and technical run-time semantics of the application. It's still a UML model, but is expressed (because of the conversion step) in a dialect (i.e. a profile) of UML that precisely mirrors technical run-time elements of the target platform. The semantics of the platform-independent original model are carried through into the platform-specific model.

The next step is to generate application code itself. For component environments, the system will have to produce many types of code and configuration files including interface files, component definition files, program code files, component configuration files, and assembly

configuration files. The more completely the platform-specific UML dialect reflects the actual platform environment, the more completely the application semantics and run-time behavior can be included in the platform-specific application model and the more complete the generated code can be. In a mature MDA environment, code generation will be substantial or, perhaps in some cases, even complete. Early versions are unlikely to provide a high degree of automatic generation, but even initial implementations will simplify development projects and represent a significant gain, on balance, for early adopters, because they will be using a consistent architecture for managing the platform-independent and platform-specific aspects of their applications [SOL00].

In order to enforce the separation of concerns between specifications and their implementations, the MDA defines two kinds of models: on one hand Platform Independent Model (PIM), which is a model of a subsystem that contains no information specific to the platform or the technology that is used to realize it; and on the other hand Platform Specific Model (PSM), which is a model of a subsystem that includes information about technology that is used in the realization of it on a specific platform, and hence possibly contains elements that are specific to the platform.

In order to successfully implement model driven solutions, both modeling language standard and tools that implement them are required.

1.5.2. Modeling Languages

Since its introduction a few years ago, the Unified Modeling Language (UML) has captured industry-wide attention for its role as a general-purpose language for modeling software systems. Although it does a good job in the early phases of development process, UML does leave some things to be desired in the system design and implementation phase because it is lacking in structural and behavior constructs [BJO00].

SDL is a Specification and Description Language standardized by the International Telecommunication Unit (ITU) in recommendation Z.100. SDL is based on communicating finite state machines. The latter, called processes in SDL, are extended by data and communicate together by exchanging asynchronous messages, called signals. These signals are stored in FIFO queues and there is one common queue for each process. A complete set of graphical and formal symbols allows the designer to describe all the dynamics of a process. Main constructs available are: sending and receiving of signals, value assignment, timer manipulation, creation and deletion of process instances, function calls.

All these features make SDL well-suited for the design of event-driven, multi-tasking and distributed systems.

SDL is often used jointly with MSC, Message Sequence Chart, which is also an ITU recommendation. Basically MSC diagrams correspond to chronologies of horizontal arrows, representing exchanged events, drawn on vertical bars, representing system's components.

SDL models the architecture and behavior of event-driven, distributed systems in real time environments. SDL has been originated as a specification language within the telecommunication industry approximately 20 years ago. Today, SDL is often used as a full-blown programming language. Although UML is headed in a similar direction, combining the

two languages provides a modeling paradigm for visual software engineering that is more robust and effective than either language alone.

UML is becoming the standard for systems definition, design and analysis. Most recent version is called UML 2 [UML20]; it merges UML 1.x with SDL/MSC and includes improved diagrams for implementing MDA.

UML 2 defines 13 basic diagram types, divided into two general sets:

Structural Modeling Diagrams

Structure diagrams define the static architecture of a model. They are used to model the 'things' that make up a model - the classes, objects, interfaces and physical components. In addition they are used to model the relationships and dependencies between elements.

- *Package diagrams* are used to divide the model into logical containers or 'packages' and describe the interactions between them at a high level
- *Class or Structural diagrams* define the basic building blocks of a model: the types, classes and general materials that are used to construct a full model
- *Object diagrams* show how instances of structural elements are related and used at run-time.
- *Composite Structure diagrams* provide a means of layering an element's structure and focusing on inner detail, construction and relationships
- *Component diagrams* are used to model higher level or more complex structures, usually built up from one or more classes, and providing a well defined interface
- *Deployment diagrams* show the physical disposition of significant artifacts within a real-world setting.

Behavioral Modeling Diagrams

Behavior diagrams capture the varieties of interaction and instantaneous state within a model as it 'executes' over time.

- *Use Case diagrams* are used to model user/system interactions. They define behavior, requirements and constraints in the form of scripts or scenarios
- *Activity diagrams* have a wide number of uses, from defining basic program flow, to capturing the decision points and actions within any generalized process
- *State Machine diagrams are* essential to understanding the instant to instant condition or "run state" of a model when it executes
- *Communication diagrams* show the network and sequence of messages or communications between objects at run-time during a collaboration instance

- *Sequence diagrams* are closely related to Communication diagrams and show the sequence of messages passed between objects using a vertical timeline

- *Timing diagrams* fuse Sequence and State diagrams to provide a view of an object's state over time and messages which modify that state

- *Interaction Overview diagrams* fuse Activity and Sequence diagrams to allow interaction fragments to be easily combined with decision points and flows.

1.5.3. Model Validation and Verification

Validation and verification are indispensable requirements for effective and successful quality management. Although this principle enjoys broad awareness, its realization in the development process is often not pursued with the appropriate persistence [STU04].

The most recurrent definition of validation and verification in SE is found in Boehm comments [BOE81]. Accordingly, validation is the evidence that the correct system has been built and that it fulfills the environment requirements. Verification means the evidence that the system has been correctly built: each partial result corresponds exactly to the specification defined in an early phase.

On one hand, validation ensures quality of the requirements and the product; while, on the other hand, verification ensures that the implementation of the requirements complies with the requirements. Thus, verification ensures the quality of the product.

Reviews and tests are the most used methods for validation and verification. Nonetheless, formal validation and verification are more effective and more elegant than reviews and tests and they permit to derive an implementation from a validated and verified formal specification and its corresponding models.

Many formal methods for validation and verification have been developed in order to help the development of complex systems. SDL has been used for many years for the formal specification of communication systems. This language permits to review the behavior at the interfaces on model level. Then, after optimization and refining, a source code is generated from model. The obtained code behaves accordingly to the specification.

UML 2.0 combines the well-known advantages of SDL and the traditional UML. The latest architecture diagrams of UML 2.0 fulfill the necessary preconditions regarding the modeling and gradual refinement of the complete system. This architecture diagram allows the static modeling of a complex system. One of the advantages of this architecture diagram (and the other UML diagrams) is that this model can be partially or completely simulated. This simulation enables a verification of the system design at a very early stage of the development process. Errors in the concept can be detected when it is still possible to remove them without spending too much time and effort. The validation is done parallel to development process, while the model of the systems is continuously adjusted with the expectations.

1.6. Chapter summary and discussion

This chapter has presented a brief survey of related work. Section 1.1 has described some fundamental concepts which will be used in the rest of this dissertation. Among these concepts, the most important are Multimedia, Quality of Service and Partial Order Quality of Service. This section has also presented a brief summary on end-to-end multimedia transmissions and has stated the necessity of a new transport protocol fulfilling the partial QoS requirements of multimedia applications. Then, section 1.2 has briefly described FPTP, a protocol which solves the problem of multimedia transport within a unicast context. Next, in section 1.3 it has been given a light survey of some multicast algorithms and recent works. It has been shown that all of them have neglected the user QoS requirements in the tree construction mechanisms. Nonetheless, some work on QoS multicast have arisen some ideas which will be taken for the solution proposed in this dissertation: the use of a per-source shortest path tree and the use of strategically placed forwarding agents. It has also be taken the ALM architecture in order to deploy this new service onto a "non-multicast-native" network; i.e. the internet. In section 1.4, it has been explained the problems of deploying new services on "closed" devices (routers, switches, etc.) and it has been exposed a technique which can solve this problem, i.e. programmable networks. This section has also described two programmable and active platforms, SARA and JavaProxy, tested in the context of two research projects: GCAP and @irs++. Finally, section 1.5 states the importance of systems modeling, validation and verification and describes a SE method created by the OMG named MDA and a language which fulfills the method requirements, UML 2.0, which combines the traditional UML 1.x features with SDL/MSC.

It is undeniable that a big number of advantages follow from the use of an adequate methodology and language for modeling, development, validation, verification and deployment. In this works it has been used MDA for system conception. It has also been used the UML 2.0 language for system analysis, design, development, validation and verification. Finally, a new generation tool called TAU Generation 2 [TAUG2] has been used in order to automate most of the processes described previously.

Next chapter describes our first outcomes on dynamic protocol deployment, and the experiments which motivated the remaining of this dissertation. It also describes a network organization model which represents at the same time the users' QoS constraints and the network performances: this model is named Hierarchized Graph. This HG will be the base for an algorithm aiming at creating a Degree-Bounded Shortest-Path-Tree which will be explained in chapter 3.

Chapter 2

A Differentiated QoS Single Source Multicast Model

This chapter is divided into two main sections.

First section describes the first single outcomes of this dissertation. These results were obtained within GCAP [GCAP] and @irs++ [@irs++] projects context. The goal was to extend FPTP unicast capabilities in order to create a single-source multimedia multicast communication. The first step was to extend the Point-to-Point unicast capabilities to Multicast-to-Multicast (MC2MC). This extension is explained in section 2.1.1. Next, the protocol was extended in order to obtain a more complex proxy architecture by creating a *multiclient proxy server*. This new architecture had permitted to relay many MC-LANs through a single level MC tree. This architecture is explained in section 2.1.2. As this single-level multicast architecture is not scalable, a new extension was proposed in order to integrate a multilevel multicast tree formed by single FPTP connections and to enhance the multicast capabilities already reached. Nevertheless, this extension implies big changes to receiving proxies, so it was proposed to reuse the already existent elements. Then, proxies server and client were put together in the same node simplifying their deployment. This architecture is explained in section 2.1.3.

Second section presents a deeper analysis of multicast related works described on chapter 1, its advantages and its lacks on user QoS requirements. This analysis is given in section 2.2.1. Then, this analysis provides a justification for a new network organization model as a hierarchized graph. This graph, described in section 2.2.2, solves the problems described in the first section. All graph behavior is expressed here in the form of algorithms. Section 2.2.3 gives the algorithm which statically creates the graph, section 2.2.4 explains the dynamic vertex insertion and section 2.2.5 describes the vertex deletion process. Finally, the hierarchized graph is modeled by using UML and SDL in section 2.2.6.

2.1. Dynamic and Programmable Protocol Deployment Experimentation

2.1.1. First FPTP enhancement: from Point-to-Point to MC-to-MC

As explained in precedent chapter, FPTP [EXP03] is a configurable and programmable protocol aiming at guaranteeing the distributed multimedia applications partial QoS requirements. FPTP is based on a *Proxy* architecture as shown in Figure 12.

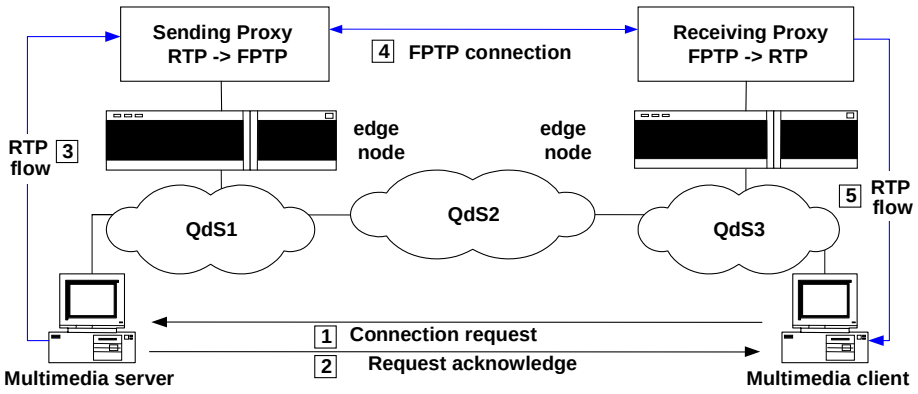

Figure 12: Unicast FPTP Proxy deployment architecture

In the context of the GCAP [GCAP] project, some test were performed relaying a sender at *Universidad Carlos III* at Madrid and a receiver at LAAS in Toulouse by using the programmable and active platform SARA [SARA]. The results were concluding and showed that traditional point-to-point multimedia connections could be enhanced by using the simple FPTP architecture [EXP02, EXP02-2].

In FPTP, the communication established from the multimedia sender to the server proxy and from the client proxy to the multimedia client uses UDP/RTP as the transport protocol and is defined to be unicast (see Figure 13).

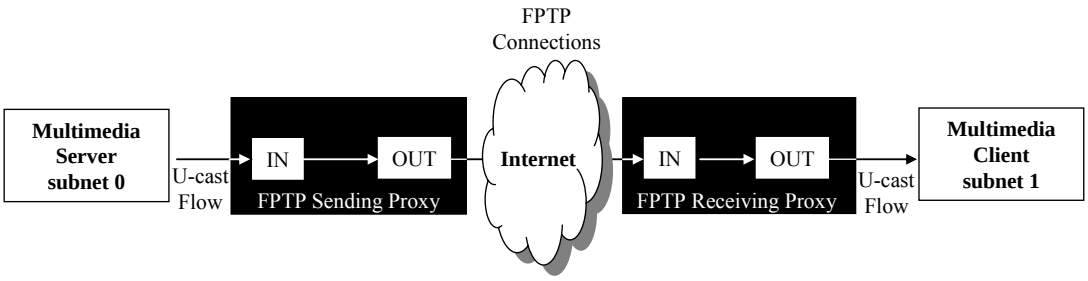

Figure 13: Point-to-point FPTP Architecture

We want to extend FPTP in order to create a multicast service. A first extension to FPTP consisted in relaying two remote local multicast networks through a FPTP link (see Figure 14). For doing so, three steps were performed:

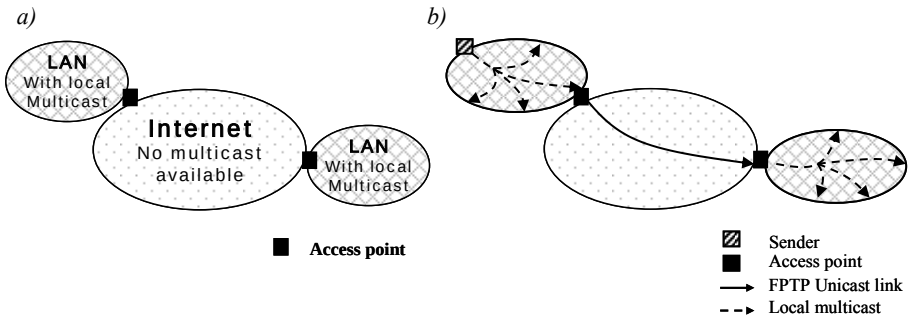

Figure 14: Relaying two remote local MC LANs by an FPTP link

- First: to change the unicast addresses from multimedia sender to server proxy into a local IP multicast address. This way, all multimedia clients within sender's LAN can receive the multimedia flow by listening to correct addresses. It is well known that traditional transport problems on WANs (loses, delay, disorder, congestion, etc.) are almost inexistent on LANs. So, no transport control mechanisms are implemented on multimedia sender's LAN

- Second: server proxy forward the received data to client proxy by using an FPTP link. FPTP will provide necessary mechanisms in order to guarantee the QoS requirements within the unreliable WAN

- Third: at multimedia client side, client proxy receives data from server proxy and forwards it to multimedia client by using a local IP multicast address. This way, all multimedia clients within the LAN can receive the multimedia flow by listening to correct addresses. Again, no transport control mechanisms are implemented on multimedia client's LAN.

Such an architecture has shown to enhance the standard FPTP connection into a simple multicast service [GCAP]. Let us call this architecture MC–to–MC FPTP This architecture is shown in Figure 15.

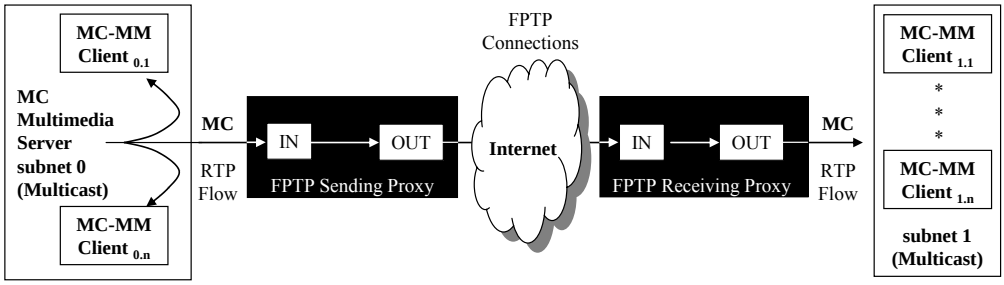

Figure 15: MC–to–MC FPTP architecture

This architecture has permitted to relay two remote local multicast networks through a partial QoS connection supported by FPTP. No changes were needed in order to obtain this architecture since local IP multicast addresses and ports behave the same manner than unicast addresses. This new configuration was tested within the context of @irs++ project [@irs++].

Then, a new extension to this architecture can be done in order to enhance the service. This new extension is shown in next section.

2.1.2. Second FPTP enhancement, P2MP: from MC-to-MC to MC-to-Multi_MC

In the precedent section, FPTP service was enhanced without changing the protocol: it was only necessary to change the unicast used addresses on sender and receiver's LAN for a local IP multicast address on both sides. This new service allowed relaying two remote single multicast networks. A natural extension can be seen as relaying many remote local multicast networks (see Figure 16). This extension implies a change to the FPTP protocol.

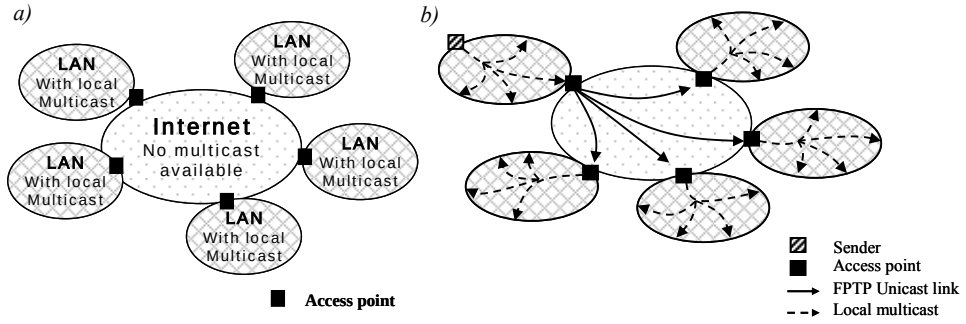

Figure 16: Relaying many local MC LANs by FPTP links

The simple FPTP server proxy was extended in order to accept many client proxies' connections. The network configuration in this case looks like Figure 16 b. Here, a FPTP sending proxy was configured for having a more evolved control module which instantiated a single output module for each connection request. Then, input module at sending proxy replicated the received packets to each output module. This way, it was possible to relay many remote local multicast LANs through single FPTP connections. As in precedent section, local unicast addresses are changed into local IP multicast ones.

This new service can be described in 4 steps:

- First, as in precedent section, the local unicast address in the sender's side is changed into a local multicast address in order to permit to all users within the same LAN to receive the data flow
- Second, the local unicast address at receiver(s) side is changed into a local multicast address
- Third, an evolved control on FPTP sending proxy waits for receiving proxies to connect. On each connection request, the control module instantiates a new output module which will receive the data flow from a common internal pipe
- Finally, each pair output-input module negotiates the required QoS constraints.

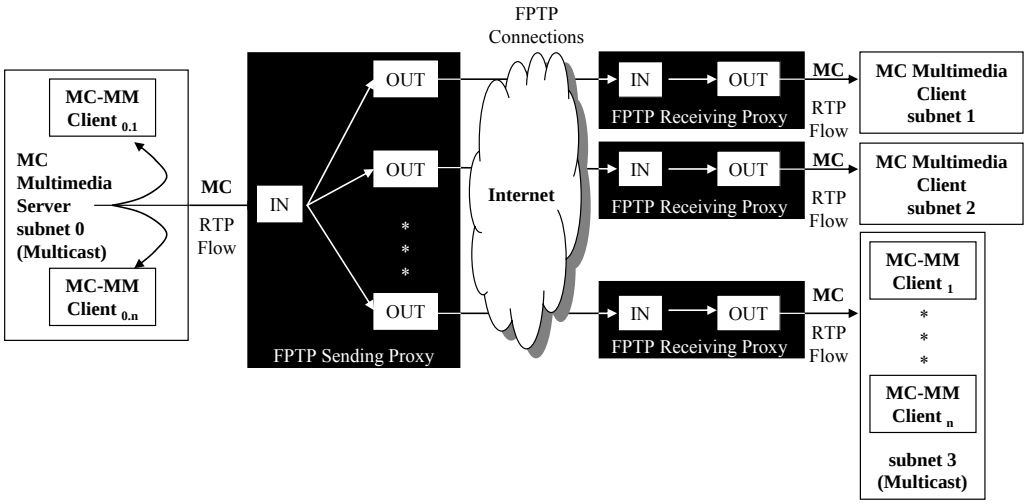

Figure 17: P2MP FPTP Architecture

It is necessary to remark that this change does not imply any change on the sender or in the receiver(s) since they still send/receive to/from a local IP multicast address.

An immense advantage of such an architecture is that, as all pairs sending_proxy_output – receiving_proxy_input modules have its own transfer control (loses, flow, partial reliability, etc.), then each connection can configure its single QoS constraints. Let us call this new configuration Point-to-MultiPoint (P2MP) FPTP Architecture (see Figure 17).

So, this new configuration relays many remote local multicast LANs through single FPTP links, each one having its own QoS constraints.

Some outcomes of this architecture can be seen in [EXP03 and GAR05].

Nevertheless, even if this new configuration greatly extends the single FPTP architecture, it creates a single level multicast tree among FPTP sending proxy and FPTP receiving proxies. This architecture allows only some receivers and it is not scalable. A multi-level multicast tree would allow a bigger number of local multicast networks to be relayed.

Next section describes a new extension solving this problem.

2.1.3. Third FPTP enhancement, Multi P2MP: Differentiated QoS Single source Multicast

As said before, a single level multicast tree is not scalable and can only support some connections. A multilevel tree allows avoiding the sending proxy overload and distributes the work on the other network elements. So, the goal of this new FPTP extension is to relay a big number of local multicast networks through a multicast tree formed by single FPTP connections. This new configuration is shown in Figure 18.

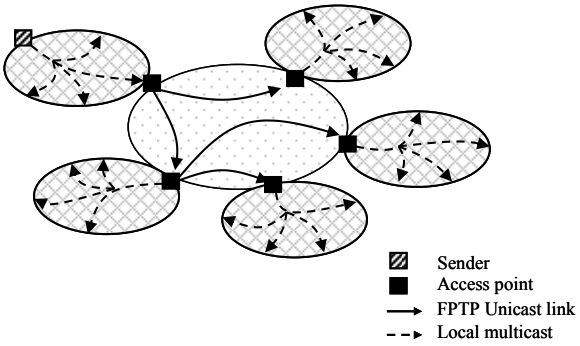

Figure 18: Multilevel Multicast FPTP

This new architecture implies to change the FPTP receiving proxy in order to allow him to forward the received data not only to a local IP multicast address, but to other FPTP receiving proxies. Let us call this new kind of element a FPTP Receiving/Sending Proxy.

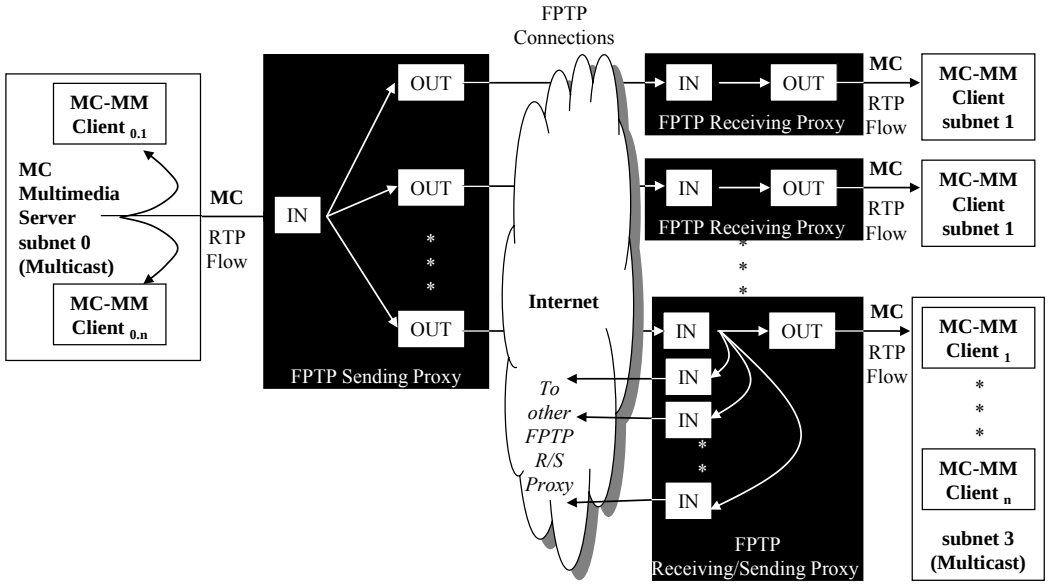

Figure 19: Differentiated QoS Single Source Multicast Architecture

In a modular FPTP proxy architecture, this change would be implemented by instantiating many output modules within the receiving proxy as shown in Figure 19. This change also implies to have a control module on each proxy in order to receive the connection requests from other proxies.

This conceptual architecture could allow obtaining a multilevel multicast tree among proxies with a different QoS configuration for each link, i.e. a Differentiated QoS Single Source Multicast Service (see Figure 19).

This architecture implies important changes to FPTP receiving proxies. An alternative solution targeting to obtain the same behavior is to reuse the FPTP proxies:

- First, just as in precedent sections, local unicast addresses on sender/receiver LANs are changed into IP local multicast addresses

- Second, again as in precedent section, the sending proxy instantiates a single output module for each receiving proxy connected to him
- Next, the receiving proxy is implemented as in the first extension; it forwards the received data to a local multicast address
- In the receiving proxy node, a sending proxy is also instantiated. This sending proxy will be in charge of managing all connection requests
- This FPTP sending proxy will not receive data from a multimedia sender but from the local multicast address used by the FPTP receiving proxy (see Figure 20).

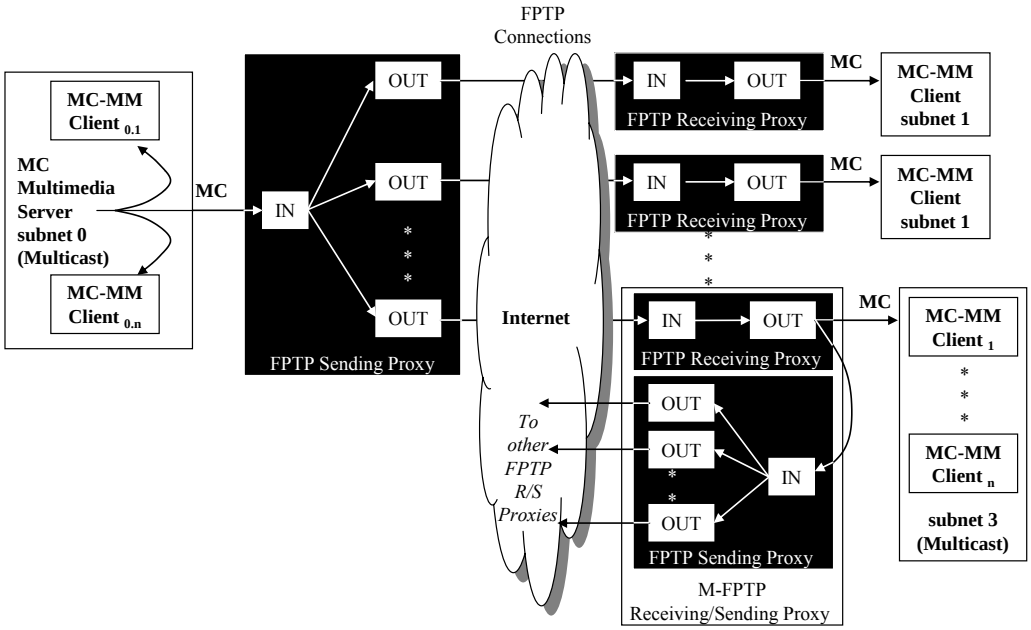

Figure 20: Differentiated QoS Single Source Multicast Architecture, modules reuse solution

This new architecture is called Multicast-FPTP (M-FPTP). M-FPTP presents many advantages:

- It does not require many modification on the FPTP receiving proxy
- For the receiving proxy instantiated on receiver's side, it is completely transparent since it still receives data from a local multicast address
- All this process is transparent for the sender and receiver(s) sides
- Each single FPTP unicast link at any tree level is independent from any other and can negotiate its own QoS requirements.

Nevertheless, it has a drawback: the multicast tree is not automatically built. The next section shows how to solve this limitation

2.2. A Hierarchized Graph

2.2.1. Introduction and Motivation

As said before, IP multicast has problems which hinder its ubiquity. ALM provides a solution. Nonetheless, ALM also presents some problems.

In a multicast session, all users cannot (or do not want to) receive the same QoS. This fact is easy to understand in the context of a service where the QoS is paid and where some users would pay for a high QoS while some others would comply with a low QoS for a lower price. Also, the users can employ access networks with different capabilities, as PDAs, GSMs, or optical fiber, providing different QoS. Nevertheless, in the general case, when using ALM, each participating member can forward data to any other member within the session. Such a service is well suited when it is established that each user will receive the same data flow. In the case of a differentiated QoS, the traditional ALM model needs to be adapted.

Another important point to be taken into account is that some users are not able to forward all possible QoS. For example, in a multimedia session, a video flow is transmitted to participants. In this session, some users cannot (or do not want to) receive more than 5 img/sec while some others can (or want to) receive 10 img/sec. It is easy to see that a user receiving 5 img/sec cannot forward data to those users asking for 10 img/sec.

As another example, if the video flow is defined to be multi-layered, some users would prefer to receive only some layers, while some other users would prefer to receive the whole set of layers. Again, it is easy to see that a user receiving only some layers, but not all of them, will not be able to forward data to those users asking for the whole layers.

The simple ALM model allows a given node *x*, receiving a low QoS, to forward data to a node *y* asking for a high QoS. In this ALM model it is necessary to send a higher QoS to node *x* so it can forward data to *y*. This fact leads to two problems: on one hand, sending more data to a user than he *asked for*, means bandwidth wasting; on the other hand, sending more data to a user than he *can* receive can lead to network congestion.

Differentiated QoS Single source Multicast service can solve the problem of sending different QoS to each node in the network. Nevertheless, as explained before, this service needs an adapted multicast tree which takes into account the different QoS requirements, i.e. to avoid a node *x* receiving a low QoS to forward data to a node *y* asking for a higher QoS.

In chapter 1, it has been given a brief description of traditional multicast algorithms; it has been described three of the most important and used algorithms: Prim, Kruskal and Dijkstra. These algorithms are only based on distances or weights and do not take into account the user's QoS requirements, thus they cannot be used to create a multicast tree among proxies in the context of a differentiated QoS single source multicast service.

In the same chapter, it was given a survey of recent work on optimizing more than one QoS criterion at the same time and on QoS multicast. Most of these work concerns the development of multicast QoS routing protocols, protocols that select multicast paths under QoS constraints. Nonetheless, the measurements taken into account to select the paths, concern only the network, i.e. bandwidth, loss ratio, cost, delay, RTT, etc, but none of them considers the user QoS.

Then, to our knowledge, there is no algorithm dealing with the problem described above.

Next section proposes a model which takes into account the network capacities and the user constraints, called Hierarchized Graph (HG). This HG represents the network as a weighted-directed graph where each vertex represents a node in the network, each directed edge represents a possible forwarding link which respects the user QoS constraints while the edge weight can represent network features as delay, cost, RTT or bandwidth, or a function of all of them. This way, the HG constrains possible forwarding paths by avoiding a highly constrained user to forward data to another user which is less constrained, i.e. a node *x* can only be served by a node *y* receiving a higher (or equal) data flow, or by the source (which has the highest QoS).

Taking this HG as the point of departure, any tree obtained from it and having the session source as the tree root, will respect the user QoS hierarchy.

2.2.2. Graph Definition

To express these user QoS and network constraints and improve the network use and session performance, it has been proposed to represent the network as a hierarchized, directed, weighted graph called simply Hierarchized Graph (HG). In this HG each hierarchy represents a user QoS; each directed, weighted edge represents a possible forward link which respects the user QoS constraints, the edge weight can represent network features as delay, cost, RTT or bandwidth, or a function of all of them and each vertex represents a node within the network.

Prior to graph definition, let us give some notations:

- Let QoS=$\{QoS_0, QoS_1, ..., QoS_k\}$ be the set of all possible user QoS requirements in the session
- Let QoS_i be "higher than" QoS_{i+1} and let us represent this property as: $QoS_i > QoS_{i+1}$ [1]
- Let QoS_0 be the maximal QoS corresponding to the QoS provided by the multimedia source.

In a formal way, the set of network nodes, user QoS constraints, possible forward links and network performances can be modeled as a directed weighted graph $\boldsymbol{G = \{V, E\}}$, a source vertex $v \in V$ and a weight function $\boldsymbol{w: E \rightarrow \mathfrak{R}^+}$ where

- $V=\{s, v_1, v_2, ..., v_n\}$ (*all the nodes within the network*)

It has been defined a function κ which maps each element in $\boldsymbol{V}$ to one element in *QoS*.

- $\kappa(v)$: $V \rightarrow QoS$

[1] For example, for a video flow, if QoS_i = 10 img/sec then QoS_{i+1} < 10 img/sec

Then, set ***E*** contains all ordered pairs (*u, v*) elements of *V* such that *u* is different from *v* and the QoS of vertex *u* is higher than or equal to the QoS of vertex *v*

- $E \subseteq V\,X\,V \mid \forall u, v \in V$, if $\{ [\kappa(u) \geq \kappa(v)] \ \wedge \ u \neq v \} \ \rightarrow \ (u, v) \in E$

It is defined a weight function ***w: E*** → $\mathfrak{R}^+$ in ***G*** which gives a positive real weight value to each edge. As said before, this weight function can be mapped to any network feature as RTT, bandwidth, cost, delay, etc. or to a function of all/some of them.

Example: given a set of network nodes *V* = {*A, B, C, D, E, F*} and source vertex *A*, if κ function is $\kappa(A) = QoS_0$, $\kappa(B) = QoS_2$, $\kappa(C) = QoS_2$, $\kappa(D) = QoS_1$, $\kappa(E) = QoS_1$, $\kappa(F) = QoS_2$ (see Figure 21 a), then E = {(A, B), (A, C), (A, D), (A, E), (A, F), (D, E), (D, B), (D, C), (D, F), (E, D), (E, B), (E, C), (E, F), (B, C), (B, F), (C, B), (C, F), (F, B), (F, C)}

The resulting Hierarchized Graph is shown in Figure 21 b (*in this figure, the edges weight is given just as an example*).

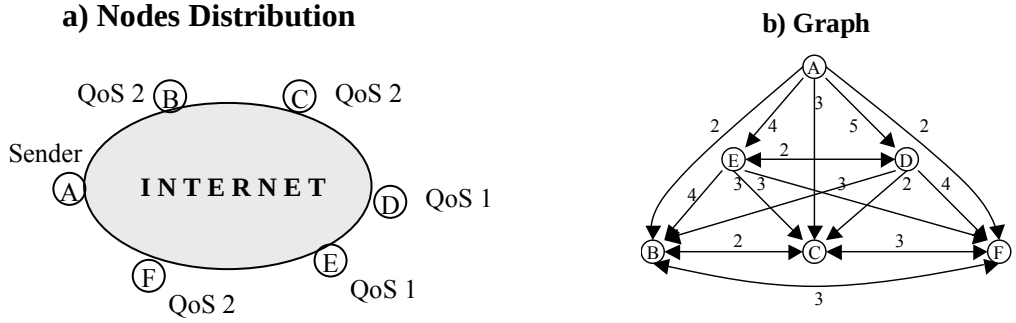

Figure 21: Hierarchized Graph example

From the definition of ***E***, it is possible to see that ***G*** is not a complete graph. The number of edges in a complete directed graph is ***n(n-1)***. The number *N* of edges in a Hierarchized Graph is

$$N = C(C-1) - \sum_{i=1}^{n} \left(C_i \sum_{j=0}^{i-1} C_j \right)$$

Where

- V_i = set of vertices with QoS_i
- C = |V|
- $C_i = |V_i|$ *($C0 = 1$ as there is only one vertex with QoS_0: the source)*
- n = |QoS|

For the graph shown in Figure 21, the number of edges is

V = {A, B, C, D, E, F}
V_0 = {A}
V_1 = {D, E}
V_2 = {B, C, F}
C = |V| = 6
$C_1 = |V_1| = 2$

$$N = 6(6-1) - \sum_{i=1}^{2} \left(C_i \sum_{j=0}^{i-1} C_j \right)$$

$$N = 30 - \left(C_1 \cdot C_0 + C_2 \left(C_0 + C_1 \right) \right)$$

$$N = 30 - \left(2 \cdot 1 + 3 \left(1 + 2 \right) \right) = 19$$

$C_2 = |V_2| = 3$
$n = 2$

It is important to remark that, as the HG expresses the user QoS constraints, any tree obtained from this HG will respect these user QoS constraints.

2.2.3. Static Graph Construction

So, the construction of an HG starts with a given set of vertices V. For each vertex $v \in V$, it is maintained an attribute $\kappa[v] \in$ QoS, which is the QoS requirement of vertex v. Set E is then constructed by following the properties given before. Algorithm 1 illustrates this process.

HG_CREATION (V)

1. $E \leftarrow \varnothing$
2. $\forall u, v \in V \mid u \neq v \wedge \kappa[u] \leq \kappa[v]$
3. $E \leftarrow E \cup \{(v, u)\}$

Algorithm 1: Hierarchized Graph creation

Lines 2 and 3 select the pair of vertices matching HG properties and add the new edge to set E.

2.2.4. Dynamic Vertex Insertion

A desired property in multimedia multicast systems is dynamism, i.e. the capacity of the system to dynamically accept nodes login/out in the course of a session while maintaining the system properties. All this can be translated into *vertex insertion/deletion* within the graph model. Algorithm 2 shows the vertex insertion procedure for an existent graph.

ADD_Vertex(v, QoS $_v$)

1. $\kappa[v] \leftarrow$ QoS v
2. $\forall u \in V$
3. if $\kappa[u] \leq \kappa[v]$
4. $E \leftarrow E \cup \{(v, u)\}$
5. if $\kappa[u] \geq \kappa[v]$
6. $E \leftarrow E \cup \{(u, v)\}$
7. $V \leftarrow V \cup \{v\}$

Algorithm 2: Dynamic vertex insertion algorithm

Line 1 updates attribute κ for vertex v. Lines 2, 3 and 4 select all vertices in V having a QoS lower than or equal to QoS of vertex v and insert edge (v, u) into set E. Lines 2, 5 and 6 select all vertices in V having a QoS higher than or equal to QoS of vertex v and insert edge (u, v) into set E. Finally, line 7 inserts vertex v into set V.

2.2.5. Vertex Deletion

The dynamic vertex deletion is an easy process; it needs to delete the desired vertex from set *V* and all edges from/to this vertex from set *E*.

Vertex_deletion (w)

1. $V \leftarrow V - \{w\}$
2. $\forall(u, v) \in E \mid (u == w) \vee (v == w)$
3. $E \leftarrow E - \{(u, v)\}$

Algorithm 3: Vertex deletion algorithm

Line 1 deletes vertex *w* from set *V* and lines 2 and 3 select and delete all edges from/to *w*.

2.2.6. Graph modeling

The Hierarchized Graph has been modeled by using UML and SDL. This section presents the diagrams corresponding to each element and operation within the graph.

In order to create a Hierarchized Graph, it has been defined a data-type QoS = {QoS_1, QoS_2, …, QoS_n} which corresponds to the user QoS constrains, and an Boolean order relation function '>' such that $QoS_i > QoS_{i+1}$ (*the inverse relation '<' is defined as* $QoS_i < QoS_{i-1}$) and QoS_0 is the maximal QoS. This QoS data-type will be related to every single vertex as a parameter (see Figure 22). As a particular case *QoS* data-type can be set as QoS 0, 1, 2, 3 and 4.

Figure 22 shows Class *Vertex* which *contains* a parameter *qos* of type *QoS,* which expresses the user QoS, and contains also an *ID*. *Vertex* class also defines operation for getting/setting its inner parameters: *getID( )* returns the *Vertex*'s ID, *getQoS( )* returns the *Vertex*'s QoS, *setQoS(QoS)* states the *Vertex*'s QoS *setV(ID, QoS)* states *Vertex*'s ID and QoS.

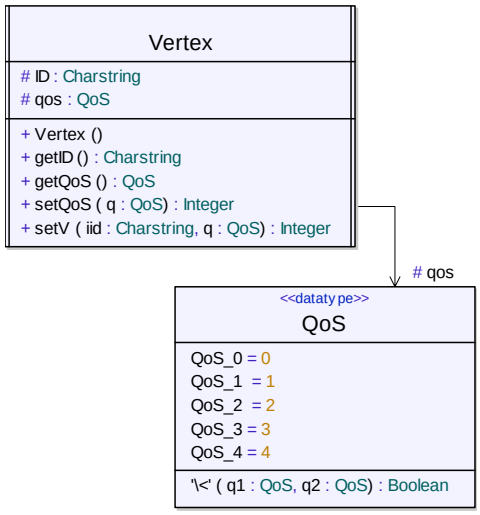

Figure 22 : UML Diagram for Classes Vertex and QoS

In the description given in precedent section, an edge could be represented by a pair $u, v \in V$. In the UML model, class *Edge* is formed by three parameters (see Figure 23):

- v1 of type String: representing element $u \in V$
- *v2* of type String: representing element $v \in V$
- *w* of type Integer: representing the edge's weight.

This class defines the necessary operations for getting/stating its parameters: getV1(), getV2(), getW( ), setW(), setEdge(String, String, Integer).

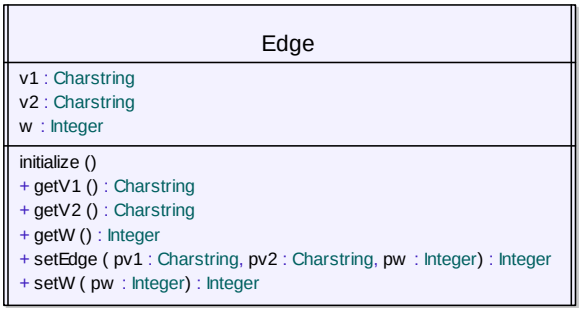

Figure 23: UML Diagram for Class Edge

Finally, it has been defined a class called *Hgraph* which represents the Hierarchized Graph. *Hgraph* contains two lists: *vList* of type *Vertex* and *EList* of type *Edge*. This class defines the operations described in precedent section: *addVertex*( ) and removeVertex( ). It also defines some accessory operations: *getNmbOfE*( ), *getNmbOfV*( ), *isPresent*( ), *getVertexAt*( ) and *findEdge*( ). Class *HGraph* and its relations with classes *Vertex*, *Edge* and *QoS* are shown in class diagram in Figure 24.

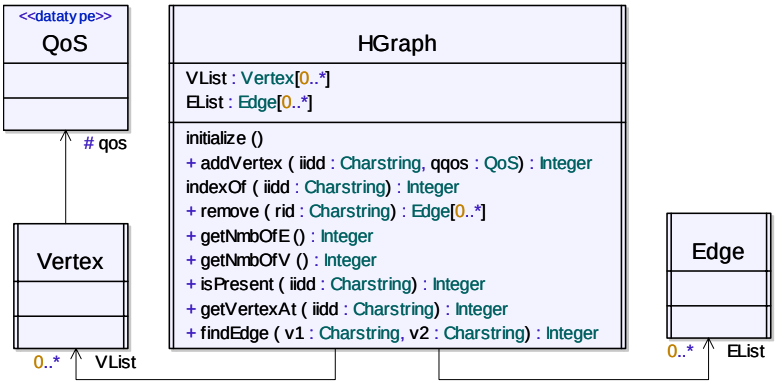

Figure 24: Class diagram for classes HGraph, Vertex, Edge and QoS

QoS is only a data type, and its only behavior is the '<' order relationship. In the case of classes *Vertex* and *Edge*, they do not have an own behavior, as they are only active by their utilization by class *HGraph*.

Class *HGraph* has its own behavior. The two main operations are *addVertex* and *remove*. *Remove* operation search and remove the selected vertex from set *V* and searches and removes all edges in set *E* to and from the selected vertex.

Operation *addVertex* is explained in details in following figures.

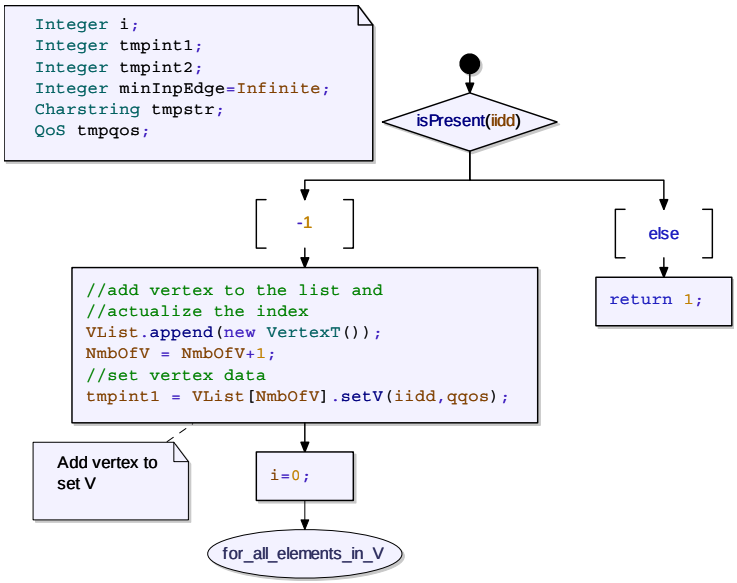

Figure 25: add vertex to set E

Figure 25 shows the beginning of *addVertex* operation. First, if the vertex to be added already exists, then operation quits. Otherwise, the vertex is added to *VList* and its parameters are updated. Then the operation searches in all elements in *VList*.

Next, for new vertex *u* and selected vertex *v*, if QoS of *u* >= QoS of *v*, then new edge (*u, v*) is added to set *E* and its weight is asked to user. Otherwise, it is verified if QoS of *u* <= QoS of *v*, and if so, new edge (*v, u*) is added to set *E* and its weight is asked to user (see Figure 26).

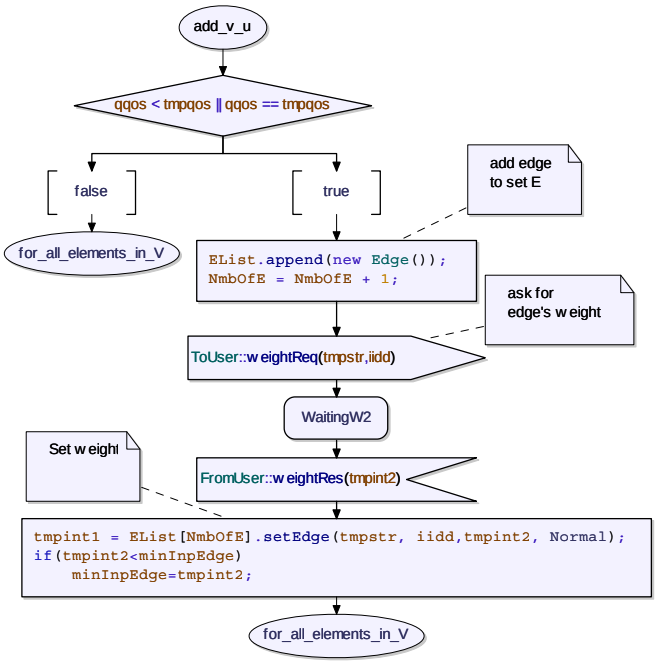

Figure 26: adding edge (v, u)

All this classes and operations will be used in next chapter in order to create a multicast tree.

2.3. Chapter summary and discussion

This chapter has shown some modifications done to FPTP in order to create a single source multicast level. First modification consisted in changing the local multicast addresses used in point-to-point FPTP in order to relay two local multicast LANs. Then, a new extension was proposed in order relay many local multicast LANs. This new extension offers the possibility of creating a single level multicast service by relaying a FPTP sending proxy and many FPTP receiving proxies. Nonetheless, this solution is not adequate for a big number of nodes. Then, a new extension to FPTP was proposed targeting the connection of a bigger number of FPTP proxies. However, this solution implies substantial changes on FPTP receiving proxies, so a solution reusing the existent elements was also proposed.

This new service has the potentiality to relay many multicast LANs through single FPTP links and to define a different QoS constraint for each one.

Such a service, based on differentiated QoS constraints, needs an adapted multicast tree. In the previous chapter, it has been given a brief survey of existing multicast algorithms and protocols. This chapter concludes that none of them fulfills the requirements of a service oriented to differentiated user QoS constraints.

Then it has been proposed an adapted network model called Hierarchized Graph which takes into account the network capacities and the user constraints. This graph solves also the problem of network resources wasting caused by the all-to-all policy in the ALM architecture. This graph was described in the form of algorithms and then modeled by using UML and SDL.

This HG reflects the differentiated user QoS constraints but no tree is yet constructed. Nevertheless, as said before, any tree (sub-graph) obtained from an HG will keep the same properties. Next chapter analyses the possible multicast tree algorithms given in the previous chapter and proposes a new one adapted to this particular problem.

Chapter 3

Degree Bounded Shortest Path Tree

Previous chapter describes some extensions to FPTP and proposes a differentiated QoS Single Source Multicast service. This new service needs new multicast mechanisms and models reflecting the differentiated user QoS constraints adapted to this architecture. It has then proposed a model called Hierarchized graph which forbids a node *x* to forward data to a node *y* asking for a higher QoS. This model economizes network resources and reduces network congestions.

Even if this HG solves some of the problems described previously, no multicast tree is yet proposed. Nevertheless, any subgraph or tree obtained from an HG will preserve its properties. This characteristic will be used in this chapter in order to propose an adapted multicast algorithm.

This chapter is divided into 3 main sections: DgB-SPT [GAR05, GAR05-2] algorithm description, algorithm model and validation, and simulation and results. Section 3.1.1 gives a brief introduction to multicast trees and problems; then section 3.1.2 explains the problems encountered when applying an algorithm such as Dijkstra's one on an HG and will show some experimental results. Next, section 3.1.3 recalls Dijkstra's algorithm while section 3.1.4 gives some modifications proposed in order to dynamically add a vertex to an SPT. After that, section 3.1.5 proposes an algorithm targeting the maximal fan-out limitation on an SPT.

The second section exposes a set of UML diagrams representing the algorithm model, its interactions with the user, its inner elements and its behavior. Then, section 3.2.4 shows a set of diagrams coming from running the UML 2.0 model. These diagrams show the algorithm validation.

The third section provides a set of measurements resulting from simulations done by implementing the algorithm in JAVA. This section explains the experiment characteristics and compares the obtained results with those obtained when no maximal fan-out limit was defined. At the end of this section some conclusions and comments are given.

Complete UML models can be found in author's web site [GAR-W]

3.1. DgB-SPT algorithm

3.1.1. Introduction

In precedent chapter it has been described a new network model called Hierarchized Graph. This HG expresses the set of network nodes, the user QoS constraints, the possible forward links and the network performances. The obtained model constrains possible forward links by avoiding a node to forward data to any other node asking for a higher QoS. Any subgraph or any tree obtained from this HG will preserve the same properties.

FPTP protocol is oriented to partial ordered and partial reliable multimedia flows. The extensions done to FPTP in precedent chapter are oriented to the same kind of data flows.

In a real-time multimedia service, the most important property to be improved is the end-to-end delay. In order to favor users' interactions, it is indeed necessary to reduce the interval of time from data production until data presentation. In a multimedia multicast service, it is needed a tree with the same en-to-end delay optimization. This kind of end-to-end delay optimized tree is called Shortest Path Tree (SPT) and the most used algorithm to obtain it is Dijkstra's one.

3.1.2. Problems with SPT based on an HG

In ALM, any single node can forward data to any other member of the multicast group. HG constrains this model by restricting multicast group members to forward data only to those members having QoS constraints lower than or equal to itself. HG model also defines a multimedia source vertex which has the maximum QoS; it means that there is a link from the source vertex to any single vertex in the group. This fact implies that it is possible that, in the SPT, all the vertices might be connected directly to the source, i.e. the source might be overloaded. This output overload on the source vertex can lead to "acknowledge implosion" problems when transposing the multicast tree to the real network. Such a tree is not scalable. For example, let us take the HG shown in Figure 21b; if we apply Dijkstra algorithm on the HG in order to obtain an SPT, then the tree shown in Figure 27 is obtained. In this SPT all vertices are connected directly to the source.

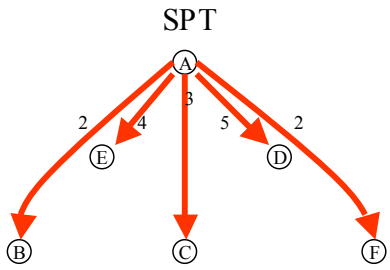

Figure 27: The SPT for the graph shown in Figure 21b

In order to verify if this source overloading is an isolated case or, instead, is a frequent case, a set of test/simulations have been performed. It has been defined a network with the following characteristics:

- The network contains 300 nodes (plus the source)
- 8 QoS levels were defined and were distributed to the vertices by using a uniform function.

It has been created an HG from this network model. Edge weights within the HG were randomly assigned from 10 to 200 by using a uniform function. After HG creation, it was used Dijkstra's algorithm in order to create a SPT which spans all the vertices from the source. This test was repeated 100 times. On each test were measured the

- Average distance from vertices to the source
- Average fan-out on the vertices
- Maximal fan-out on the tree
- Maximal distance to the source on the tree.

These tests show that the fan-out is quite small on most of the vertices:

- 75% of vertices have FO=0
- 90% have fan-out lower than or equal to 2 (see Figure 29).

Nevertheless, in the other extreme:

- 0.6% of vertices have a fan-out of at least 20 and
- 2.4% of vertices have a fan-out of at least 10 (see Figure 29).

Concerning the source

- it is never lower than 32
- it has always the maximal fan-out
- maximal fan-out in the tree can grow until 59 (see Figure 28).

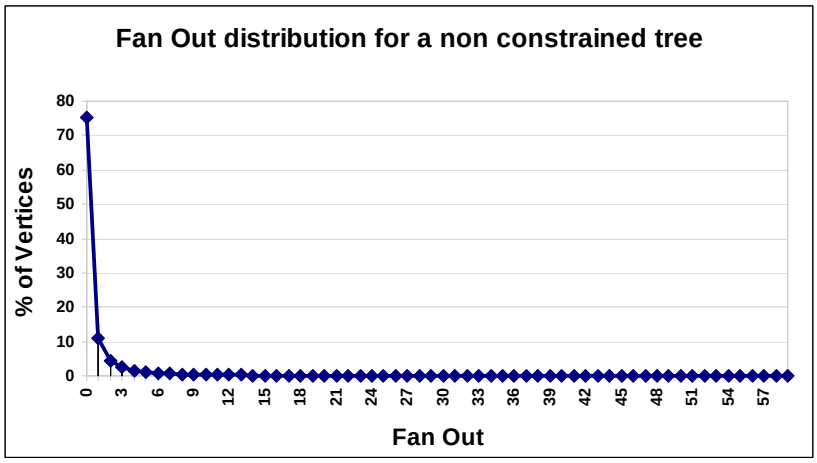

Figure 28: Fan-Out distribution for a non-constrained tree

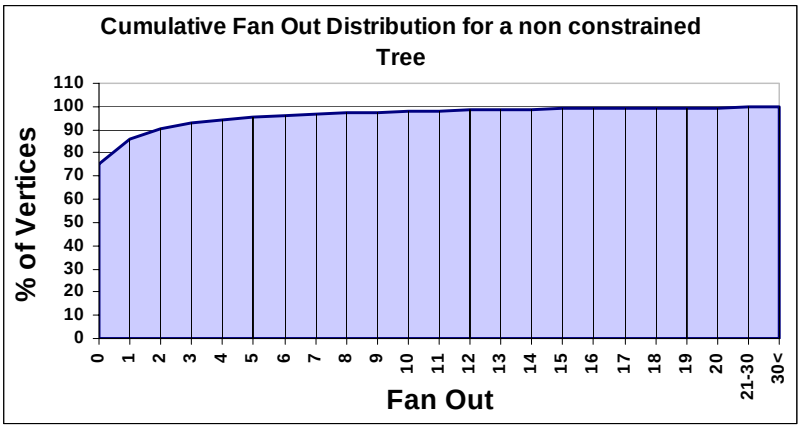

Figure 29: Cumulative Fan-Out distribution for a non-constrained tree

Concerning the distance to the source:

- Even if edge weights were uniformly distributed from 10 to 200, the mean distance to the source is considerably low (27.962) (see Figure 30)
- Maximal distance to the source is 90
- 90% of vertices have distance to source lower than or equal to 40 (see Figure 31) and
- 98.4% have distance lower than or equal to 50.

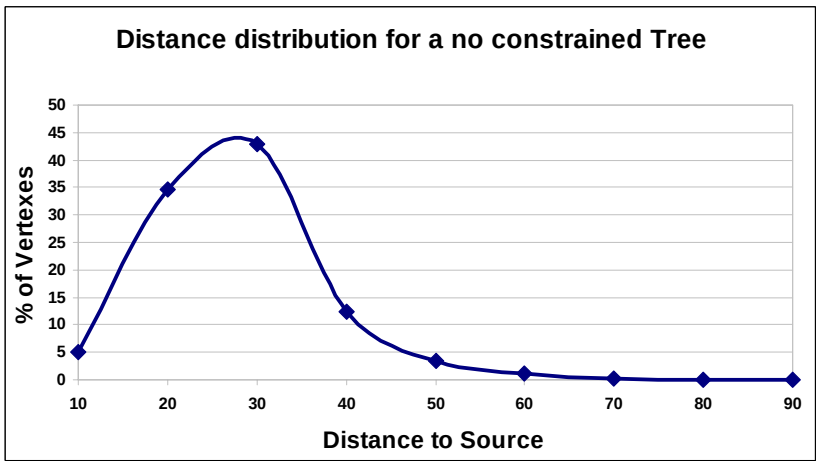

Figure 30: Distance distribution for a no constrained Tree

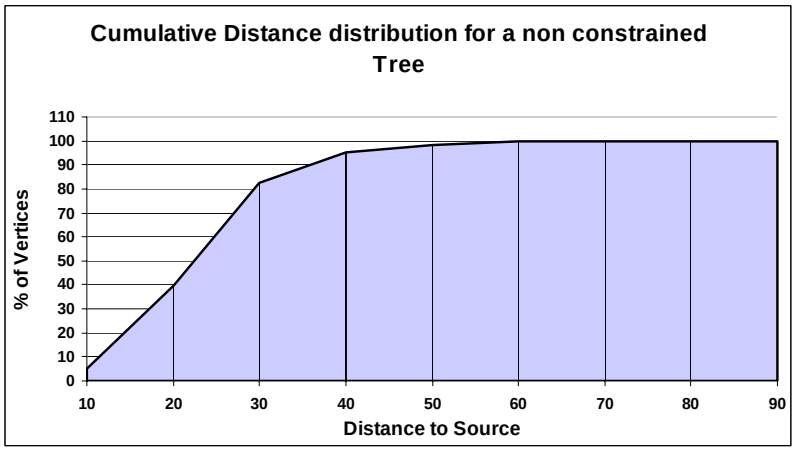

Figure 31: Cumulative Distance Distribution for a no constrained Tree

It is possible to conclude from the performed tests that, applying Dijkstra Algorithm on an HG obtained from the network model showed before:

- Issues a SPT which respects the user QoS constraints
- The obtained SPT performs well concerning the distance to the source
- There are 2.4% of vertices, including the source, which are overloaded with a fan-out of at least 10
- The source is the most overloaded vertex and its fan-out is always between 32 and 59.

In order to solve this output overloading problem at the source it is necessary to limit the maximal output degree on the vertices within the spanning tree. Let us analyze this maximal output degree constrained tree and the possible ways to obtain it.

A spanning tree where the maximal output degree of any node in the network is limited is called Degree Bounded Tree (DBT), but such a tree cannot guarantee shortest-path properties. Let us take the graph shown in Figure 21 as example and let us define the maximal output degree of any vertex as 2. Then, Figure 32 shows one of possible DBT fulfilling this maximal output degree. Comparing the two trees, it is possible to observe:

- A DBT is not always a SPT
- An SPT might have overloading problems
- A DBT has not as good performance as the SPT concerning the end-to-end delay (there exists shorter paths to vertices B, C and D), but
- A DBT does not have the overloading problem of SPTs.

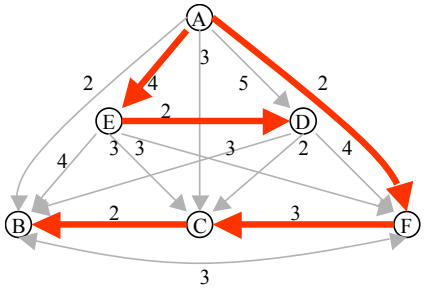

Figure 32: A DBT for the graph shown in Figure 21

It is so desired to keep the SPT properties while constraining the maximal output degree on all vertices within the tree. The expected tree will not perform as well as a SPT concerning the end-to-end delay but it will not have the overloading problems of it. Let us call this tree **Deg**ree **B**ounded-**S**hortest **P**ath **T**ree (DgB-SPT).

As explained before, many research work have dealt with optimizing more than one cost criterion simultaneously and with QoS multicast. Nonetheless, to our current knowledge, none of these works are adapted to our problem. The difference with other works is the type of graph serving as source of the tree.

Ravi et al [RAV93] take into account an undirected graph and their goal is to span a subset of the entire graph. Nonetheless, an HG is a directed graph and our goal is to span the entire set of vertices.

Radha et al [RAD01] *minimize* the fan-out degree for each vertex based on a non-weighted graph. In an HG, the minimal degree for each vertex is 1, and this is equivalent to find a Hamiltonian path. Even more, for the problem treated in this work, it is not desirable to reduce the output degree to the minimal, but only bound it to a defined limit (FO) in order to avoid overloading problems.

Ito et al have done a remarkable work in [ITO02]; however, even if this work approaches to our objectives, their algorithms are based on acyclic graphs, and an HG is not always acyclic.

Another interesting work is the Degree-Constrained Steiner Tree Problem [VOS92] defined as: let G = (V, E) be an undirected edge weighted graph. The *degree constrained Steiner problem* (DCSP) is: find a minimum weight subgraph connecting the vertices of a stated subset $Q \subseteq V$ of vertices such that for each vertex $i \in Q$ un upper bound $d(i)$ for the number of edges incident to i is not exceeded. DCSP includes the degree constrained minimum spanning tree problem as the specified subset may become the whole vertex set. Nevertheless, this work is based on undirected graphs. Another aspect of this work which avoids its utilization for an HG is that we are interested in a rooted tree, and the DCSP does not take into account this.

Let us analyze how to obtain a DgB-SPT.

3.1.2.1. *Lightest Degree Bounded Tree*

The surest way to obtain a DBT which minimizes the end-to-end delay (a DgB-SPT) is to calculate all possible Degree Bounded Trees and to choose the “lightest” of them. Nevertheless, the number of trees to be calculated is enormous and so, this method is impractical. For example, let us take the graph shown in Figure 21 as reference and let us limit the maximal

output degree to 3 (it means that all configurations from 1 to 3 fulfill the limitations). For this example there exist approximately 290 DBTs fulfilling the maximal output degree limit. Figure 33 shows the three lightest trees. This figure shows that DBT_4 is the lightest DBT. This tree does not have the same performance than the SPT (Figure 27) concerning the end-to-end delay but it does not have the overloading problems of it.

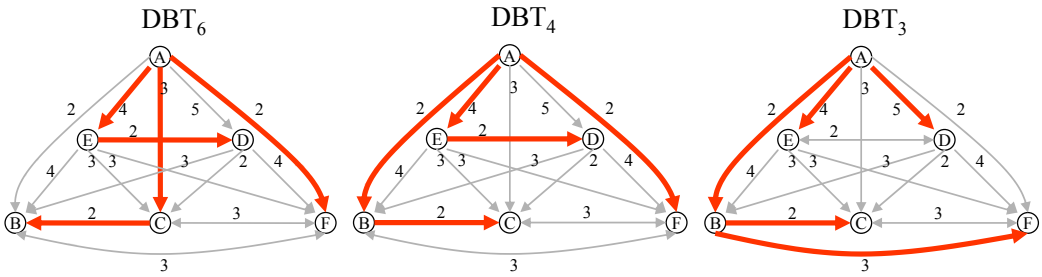

Figure 33: The three lightest DBTs

This method is the surest way to find the lightest DBT since it explores all possible DBTs. This method can work well on little graphs but it is not scalable to big ones.

It is possible to think about an iterative method: the heavier edge in the graph is pruned at each iteration; then Dijkstra's algorithm is applied on the new graph. These iterations are repeated until the desired tree is obtained.

This method might have a better performance than the precedent one; nevertheless, it is inexact because it does not ensure the desired graph, and the iterative application of Dijkstra's algorithm can lead to non performing results.

In order to obtain a near-SPT tree without output overloading from an HG, next section proposes a modification of Dijkstra's algorithm where the maximal output degree has been constrained; this new algorithm is called Degree Bounded Shortest Path Tree (DgB-SPT). To facilitate the understanding of these modifications, it will be necessary to the recall how Dijkstra's algorithm works.

3.1.3. Dijkstra Algorithm

Shortest-path algorithms typically exploit the property that a shortest path between two vertices contains other shortest paths within it. The following lemma and its corollary state the optimal substructure property of shortest paths more precisely [COR90].

Lemma 1 (Sub-paths of shortest paths are shortest paths). Given a weighted, directed graph $G = (V, E)$ with weight function $w : E \rightarrow \Re$,

- let $p=\{v_1, v_2, ..., v_k\}$ be a shortest path from vertex v_1 to vertex v_k and
- let $\delta(v_1, v_k)$ indicate the weight of path p (*the shortest distance from vertex v_1 to vertex v_k in G*) and,

- let $p_{ij}=\{v_i, v_{i+1}, ..., v_j\}$ be the sub-path of p from vertex v_i to vertex v_j for any i and j such that $1 \le i \le j \le k$

- Then p_{ij} is the shortest path from v_i to v_j.

Corollary 1. Let $G = (V, E)$ be a weighted, directed graph with weight function $w : E \rightarrow \Re$. Suppose that a shortest path p from a source s to a vertex v can be decomposed into $s \sim^{p'} \sim> u \rightarrow v$ for some vertex u and path p'. Then, the weight of a shortest path from s to v is $\delta(s, v) = \delta(s, u) + w(u, v)$.

The next lemma gives an extension of lemma 1 as a simple but useful property of shortest-path weights.

Lemma 2. Let $G = (V, E)$ be a weighted, directed graph with weight function $w : E \rightarrow \Re$ and source vertex s. Then, for all edges $(u, v) \in E$, we have $\delta(s, v) \le \delta(s, u) + w(u, v)$

3.1.3.1. *Relaxation*

The algorithm explained here uses the technique of relaxation [COR90]. For each vertex $v \in V$, it is maintained an attribute $d[v]$, which is an upper bound on the weight of a shortest path from source s to v. We call $d[v]$ a ***shortest-path estimate***. It is also maintained an attribute $\pi[v]$, which is the predecessor field. The shortest-path and predecessors are initialized by the following procedure.

INITIALIZE-SINGLE-SOURCE (G, s)

1. $\forall v \in V[G]$ **do**
2. $d[v] \leftarrow \infty$
3. $\pi[v] \leftarrow$ NIL
4. $d[s] \leftarrow 0$

Algorithm 4: Initializing a single source graph

After initialization, $\pi[v] \leftarrow NIL$ for all $v \in V[G]$, $d[v] = 0$ for $v = s$, and $d[v] = \infty$ for $v \in V - \{s\}$.

The process of relaxing an edge (u, v) consists of testing whether it is possible to improve the shortest path to v found so far by going through u and, if so, updating $d[v]$ and $\pi[v]$. A relaxation step may decrease the value of the shortest-path estimate $d[v]$ and update v's predecessor field $\pi[v]$. The following code performs a relaxation step on edge (u, v).

RELAX (u, v)

1. **If** $d[v] > d[u] + w(u, v)$ **then**
2. $d[v] \leftarrow d[u] + w(u, v)$
3. $\pi[v] \leftarrow u$

Algorithm 5: Edge relaxation.

3.1.3.2. *The algorithm*

Dijkstra's algorithm solves the single-source shortest-path problem on a weighted, directed graph $G = (V, E)$ for the case in which all edge weights are nonnegative.

Dijkstra's algorithm maintains a set S of vertices whose final shortest-path weights from the source s have already been determined. The algorithm repeatedly selects the vertex $u \in V - S$ with the minimum shortest-path estimate, inserts u into S, and relaxes all edges leaving u (output neighbors). In the following implementation, a priority queue Q is maintained, which contains all vertices in $V - S$, keyed by their d values.

DIJKSTRA (G, s)

1. Initialize-single-source (G, s)
2. $S \leftarrow \varnothing$
3. $Q \leftarrow V[G]$
4. **While** $Q \neq \varnothing$ **do**
5. $u \leftarrow$ Extract-min (Q)
6. $S \leftarrow S \cup \{u\}$
7. $\forall v \in Adj[u]$ **do**
8. Relax (u, v)

Algorithm 6: Dijkstra algorithm.

The following section explains how shortest path properties are used to extend Dijkstra's algorithm to dynamically add new vertices.

3.1.4. Dynamic Vertex Insertion

Algorithm 2 (pp. 45) dynamically adds a vertex v to an already created HG. Now let us suppose that an SPT is created from a given HG; then the addition of a new vertex by applying Algorithm 2 does not change the structure of SPT (*$d[u]$, $\pi[u]$ and $\kappa[u]$ attributes for all vertices, but the new one, are not modified)*.

From Algorithm 6, it is possible to affirm that, after application of Dijkstra's algorithm, all nodes in the SPT know its $d[v]$ and $\pi(v)$, *i.e. the distance to the source and parent vertex*. A new vertex v' joining the graph by using Algorithm 2 would have $d[v'] = \infty$ and $\pi(v') =$ NIL.

Then, in order to dynamically add a new vertex v' to an existent SPT (*after its dynamical addition to HG by using Algorithm 2*) it is possible to simply relax all its input and output edges within the graph. The explanation of this process depends on Dijkstra's algorithm properties and it is explained as follows.

3.1.4.1. *Input relaxation*

Let us recall that the process of relaxing an edge *(u, v')* consists of testing whether it is possible to improve the shortest path to *v'* found so far by going through *u* and, if so, updating $d[v'] \leftarrow d[u] + w(u, v')$ and $\pi[v'] \leftarrow u$.

Lemma 2 can be reformulated by:

Given a Hierarchized Graph $G = (V, E)$ with weight function $w : E \rightarrow \Re$ and source vertex *s*, and given a SPT calculated from *G*. Then, for all edges $(u, v') \in E$, we have $\delta(s, v') = \min (\delta(s, u) + w(u, v'))$.

This reformulation leads us to conclude that:

- Given a Hierarchized Graph $G = (V, E)$ with weight function $w : E \rightarrow \Re$ and a source vertex *s*, and
- Given a SPT calculated from *G*, and
- After execution of Algorithm 2 in order to add a new vertex *v'* to G
- Then, after relaxing all edges linked to *v'*, $d[v'] = \delta(s, v')$ and $\pi(v') \neq$ NIL.

3.1.4.2. *Output relaxation*

Nevertheless, it is possible that, in the new graph, *v'* is included in the shortest-path for some other vertices. Let us reformulate the process of edge relaxation by saying that: the process of relaxing an edge *(v', u)* consists of testing whether it is possible to improve the shortest path to *u* found so far by going through *v'* and, if so, updating $d[u] \leftarrow d[v'] + w(v', u)$ and $\pi[u] \leftarrow v'$; otherwise, $d[u]$ and $\pi[u]$ remain unchanged.

This reformulation leads us to assert that:

- Given a Hierarchized Graph $G = (V,E)$ with weight function $w : E \rightarrow \Re$ and source vertex *s*, and
- Given a SPT calculated from *G*, and
- After execution of Algorithm 2 in order to add a new vertex *v'* to G and
- After relaxation of all edges arriving to new vertex *v'* and
- After relaxation of all edges leaving from new vertex *v'*
- Then, $d[u] = \delta(s, u)$ for all $u \in V$.

The following algorithm recapitulates the two processes just explained.

3.1.4.3. *Dynamically adding a vertex to an SPT*

Add-vertex-to-Graph-and-SPT (*v', QoS_i)*

1. Add - Vertex (*v', QoS_i*)
2. ∀ *u* | *(u, v')*∈ *E*
3. **do** Relax (*u, v'*)
4. ∀ *u* | *(v', u)* ∈ *E*
5. **do** Relax (*v', u*)

Algorithm 7: Inserting a new vertex to a hierarchized graph and SPT

Line 1 adds the vertex to the HG; lines 2 and 3 relax all input edges and finally, lines 4 and 5 relax all output edges. It is necessary to say that, for every vertex decreasing its distance to the source, it is needed to update the distance for its entire sub tree and it is necessary to recursively relax their output edges. This operation is done by the *deepRelax* function referenced in section 3.2.3.

At this point, it has been explained the necessary modifications to Dijkstra's algorithm to dynamically add new vertices to the SPT. Nonetheless, the output overloading problem still subsists. In next subsection it is given an explanation of the way a vertex and one of its edges will be selected to be pruned to fulfill the Fan Out constraints.

3.1.5. Tree Pruning

3.1.5.1. *Selecting the vertex and edge to be pruned*

In order to fulfill the Fan Out constraints, it is necessary to prune one or more edges (one by one) from the vertex which is the most overloaded. To select this *most-overloaded* vertex it will be defined a new attribute ***φ[v]*** as the current fan out of vertex *v*. It is also defined $\mathbf{MaxOD_T}$ = *max*(*φ [v_i]*) | $v_i \in V$, and **FO** as the maximal fan-out limit to be respected by all vertices in the SPT. In order to compare the performance among two trees, let us also define **μDtS** as the mean distance from all vertices in the tree to the source. So, finding this *most-overloaded* vertex can be done by

- If $\text{MaxOD}_T > \text{FO}$ then
- find the vertex *v* for which ***φ[v]*** = MaxOD_T

Once selected this *most-overloaded* vertex *v* one of its edges must be selected to be pruned. The selection of the most appropriate edge to be pruned is not as simple as "*select the heaviest output edge*"; a bad selection can lead to very heavy trees. Let us take the tree shown in Figure 34a as an example and let us define the maximal output degree limit FO=2. Figure 34a shows a SPT having a mean distance of 10 and maximal output degree of 3. In this tree, the vertex having φ[v] = MaxOD_T is vertex *A*, and its heavier output edge is (A, C). If edge (A, C) is pruned, then vertex *C* will be reconnected by using edge (B, C) and then the weight of the tree will increment until 20 (see Figure 34b). On the contrary, if the pruned edge is edge (A, D), vertex *D* will be reconnected by using edge (B, D) and then the weight of the tree will increment only until 12.5 (see Figure 34c).

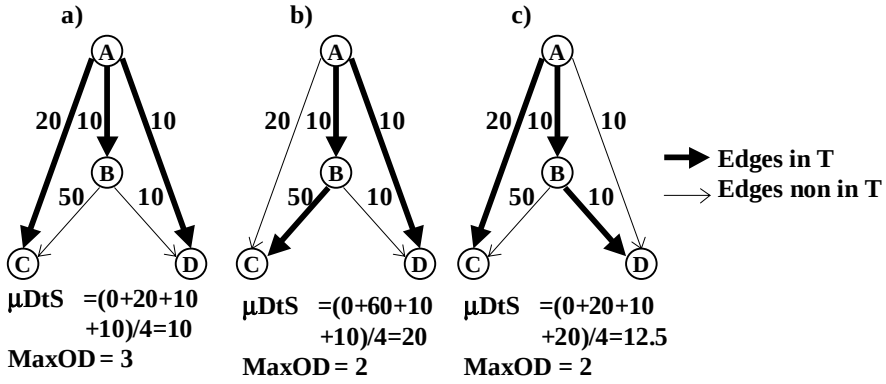

Figure 34: Differences in selecting edge to prune.

Even if both trees (Figure 34 b and c) accomplish FO restrictions, the selection of the correct edge to be pruned can drive to very different weights. So, pruning the heavier edge is not always the best solution.

A better solution comes when answering the following question: *which is the vertex which increments the less the weight of the tree when reconnecting it through an alternative path after pruning it?*

Algorithm 8 selects the vertex which increments the less the weight of the tree when reconnecting it through an alternative path after pruning and selects the edge to be pruned. Prior the algorithm, some parameters are defined:

- *alternativeDistance(v)* : the minimal distance of all alternative *s~v* paths but the one currently used in the tree
- *reconnectingCost(v)* : the difference between *alternativeDistance(v)* and *d[v]*
- *OV* : the overloaded vertex; i.e. $OV \in V \mid \varphi [OV] = \text{MaxOD}_T$
- *minimalReconnectingCost* : min(*reconnectingCost(v)*) $\forall v \mid \pi[v] = OV$.

Select-Edge-To-Be-Pruned

1. *vertexToBePrunned* ← NULL
2. *minimalReconnectingCost* ← ∞
3. *newFather* ← NULL
4. *edgeToBePruned* ← NULL
5. *OV* ← $w \mid \varphi[w] = \text{MaxOD}_T$
6. $\forall v \in V \mid \pi[v] = OV$
7. *alternativeDistance(v)* ← ∞
8. $\forall (u, v) \in E \mid u \neq OV$
9. **if** *alternativeDistance(v)* > *d[u]* + *w(u, v)* **then**
10. *selectedEdge* ← *(u, v)*
11. *alternativeDistance(v)* ← *d[u]* + *w(u, v)*

12. *reconnectingCost(v)* ← *alternativeDistance(v) – d[v]*
13. **if** *reconnectingCost(v)* < *minimalReconnectingCost* **then**
14. *minimalReconnectingCost* ← *reconnectingCost(v)*
15. *vertexToBePruned* ← *v*
16. *edgeToBePruned* ← *selectedEdge*
17. *newFather* ← *edgeToBePruned.u*

Algorithm 8: Selecting the Edge to be pruned

After initializing the variables (lines 1-4), the algorithm finds the overloaded vertex (OV) in line 5. Then, for every OV's children (line 6), the algorithm initializes an *alternative distance* variable (line 7), selects all its input edges but the actually used (line 8) and searches for the minimal alternative distance (line 9) and stores this possible *edge to be pruned* and *alternative distance* (lines 10, 11). Then, the *reconnecting cost* is calculated (line 12). Finally, the algorithm selects the *minimal reconnecting cost* (line13) and updates the variables *minimal reconnecting cost* (line 14), *vertex to be pruned* (line 15), *edge to be pruned* (line 16) and *new father* for the *vertex to be pruned* (line 17).

3.1.5.2. *Edge pruning and tree updating*

After applying Algorithm 8, a vertex ***v*** and an edge ***e*** within the tree are selected. The process of pruning consists on changing $\pi[v]$ and updating $d[v]$, i.e. assigning an alternative path and distance to vertex v; furthermore, it is also necessary to "erase" the ancient used edge e within the tree. For doing this, edges will be labeled by using a new parameter. Let $\boldsymbol{\lambda[e]}$ be a label assigned to an edge $e \in E$. This label can be Normal, Deleted or Indispensable and it is used as follows:

- All edges are initialized as **Normal**
- A pruned edge is labeled as **Deleted** (this label helps to avoid trying to use a deleted edge)
- If a vertex has an only non-Deleted input edge, this edge is labeled **Indispensable** (this is useful to avoid trying to prune an edge which would drive to a non-connected graph).

So, the process of pruning an edge consists in labeling *edgeToBePruned* as *Deleted* and then updating the distance and the predecessor of the pruned vertex. This process is shown in Algorithm 9.

Prune-vertex

1. $\pi[vertexToBePrunned]$ ← *newFather*
2. *d[vertexToBePrunned*] ← *d[vertexToBePruned]* + *minimalReconnectingCost*
3. $\lambda[edgeToBePruned]$ ← Deleted

Algorithm 9: Pruning a vertex

Nevertheless, the process of pruning an edge *(u, v)* implies to increment *d[v]* and so, increment the distance to the source for *v's* subtree.

Another consequence of pruning edge *(u, v)* is that, as *v* has incremented its distance to the source, then it is possible that *v* is no longer part of the shortest path for its subtree. So, the goal now is to find a new shortest path for the entire *v's* subtree. Algorithm 10 is a continuation of Algorithm 8 and Algorithm 9.

Subtree-Update

1. $subtree \leftarrow \emptyset$
2. $subtree \leftarrow subtree \cup vertexToBePruned$
3. $\forall v \mid \pi[v] \in subtree$
4. $\quad subtree \leftarrow subtree \cup v$
5. $\quad d[v] \leftarrow d[v] + minimalReconnectingCost$
6. $subtree \leftarrow subtree - \{ vertexToBePruned\}$
7. $\forall v \in subtree$
8. $\quad \forall (u, v) \in E \mid \lambda[(u, v)] \neq$ Deleted
9. $\quad\quad relax(u, v)$

Algorithm 10: Subtree updating

Line 1 initializes a list which will contain *v's* subtree and line 2 inserts the pruned/reconnected vertex to the list. Lines 3, 4 add *v's* subtree to the list, while line 5 updates its distance to the source. Line 6 deletes the first element, the pruned vertex, because its parent has already been updated. Finally, lines 7 to 9 update the parent and distance fields for each member of the subtree. This updating is done by using the *relax* process. As explained before, the process of relaxing all input edges of a vertex within an existent SPT updates its parent and distance to the source.

It is important to notice that:

- The relax process cannot take the deleted edges into account
- The output edges are not being relaxed. Let us remember that the process of relaxing an edge *(u, v)* consists of testing whether it is possible to improve the shortest path to *v* found so far by going through *u*. But, in this case, as d[v] is incremented, it is sure that the shortest path to any vertex cannot be improved by going through a vertex whose distance has increased.

At this point Algorithm 7 dynamically adds a vertex to an HG and to the tree, Algorithm 8 selects the vertex not accomplishing the FO constraints and the vertex/edge to be pruned, Algorithm 9 prunes the selected vertex and updates its fields and Algorithm 10 updates its subtree. The iterated execution of Algorithm 8, Algorithm 9 and Algorithm 10 leads to a DgB-SPT, i.e. a tree having Maximal Output Degree lower than or equal to a defined maximal limit FO while minimizing the tree's weight (μDtS_T).

3.1.6. Vertex Deletion

A possible solution for deleting a vertex *v* from an SPT is to connect all *v's* children to *π[v]*. This action ensures a fast reconnection of *v's* subtree and the distance to the source for all reconnected vertices would decrease. Nevertheless, this procedure could overload *π[v]* which probably was balanced. In this case it would be necessary to re-start the pruning process and prune a number of vertices equal to v's children and, as consequence, it would be necessary to update all the subtrees.

It is necessary to find a compromise between easiness and performance. The deletion process can be achieved by an easy, but non performing, method. In order to remove a given vertex *v*, *i*t is possible to remove all vertices (and its corresponding edges) from the graph and re-insert them one by one to the HG and the SPT. Even if this method is not as performing as desired, it is easy to implement by reusing the already implemented insertion process.

3.2. Algorithm Model and Validation

As problems and solutions become more complex, it is hard to describe them, and hard to understand the descriptions. An easy and powerful way to describe both problem and solutions, which can provide a good level of abstraction, allowing the architects focus on architecture, designers focus on design (rather than worrying about implementation too soon) is necessary.

As explained before, MDA and UML have been created by OMG in order to solve these problems. Unified Modeling Language (UML) is a non-proprietary, third generation modeling and specification language. The Unified Modeling Language is an open method used to specify, visualize, construct and document the artifacts of an object-oriented software-intensive system under development. The UML represents a compilation of "best engineering practices" which have proven successful in modeling large, complex systems, especially at the architectural level.

The language must be supported by a powerful tool that faithfully and efficiently implements the language, so that it can automate the mapping transformations across the various models. TAU Generation2 [TELLO] is a family of model-centric and role based tools that are amongst the first to implement the recently adopted UML 2.0 standard.

Next sections show the models describing the behavior of DgB-SPT.

3.2.1. Simple system view

The use case diagram (Figure 35) expresses the possible interactions between the user and the algorithm and between inner system elements. This diagram shows that a user will see the system as a black box where he can perform only 5 actions: Reset the system, Add a vertex, Remove a vertex, Get current tree and graph. In this diagram it is possible to appreciate that *removing* method *includes* the *adding* method.

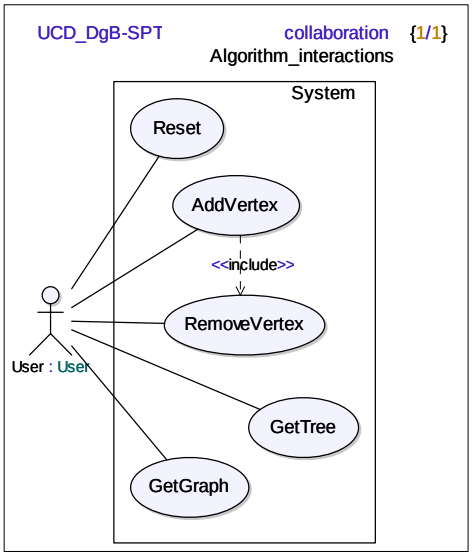

Figure 35: Use case diagram for DgB-SPT

These interactions are performed in the form of signals going from the user to the algorithm through a port, and vice versa. Some of these signals correspond to processes or operation calls. Figure 36 shows the algorithm black-box and its interfaces, and then Figure 37 shows the signals corresponding to each interface.

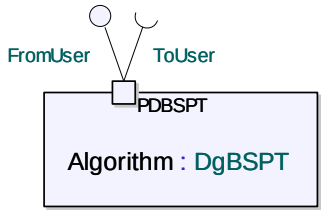

Figure 36: The algorithm and its interfaces

<<interface>> FromUser	<<interface>> ToUser
signal Reset () signal w eightRes (w eight : Integer) addTVertex (Charstring, QoS, CalculationType) : Integer setSource (Charstring) : Integer setWe (Charstring, Charstring, Integer) : Integer relax (Charstring, Charstring) : Integer +setFanOut(Integer) : Integer removeVT(Charstring):Integer	signal response () signal w eightReq(u : Charstring, v : Charstring)

Figure 37: Interfaces description for the algorithm

In Figure 37 it is possible to see that the interface *FromUser* contains the signals and the procedure calls used to interact with the algorithm and the answers given by it:

- *FromUser::addVertex(Charstring, QoS, CalculationType)*. This operation adds a vertex to the graph and the tree
    - *Charstrig* is the vertex's ID
    - *QoS* is the quality of service level assigned to the vertex
    - *CalculationType* defines if the vertex is to be added with Dijkstra or DgB-SPT algorithm.
- *ToUser::weightReq(Charstring, Charstring)*. When the vertex is added to the graph, it asks the weight of all edges added. The two *Charstring* parameters are the two vertices forming the edge
- *FromUser::weightRes(Integer)*. Answer to *weightReq()* message. It contains an integer corresponding to the asked edge's weight
- *FromUser::setSource(Charstring)*. This procedure sets a vertex as the session source, i.e. it sets vertex ancestor to *NULL* and vertex distance to source to 0
- *FromUser::setWe(Charstring, Charstring, Integer)*. This operation allows setting the weight of an defined edge
- *FromUser::relax(Charstring, Charstring)*. Applies the *relaxation* operation on a defined edge
- *FromUser::removeVT(Charstring)*. Removes a given vertex from graph and tree
- *FromUser::setFanOut(Integer)*. Sets the maximal output-degree limit to be fulfilled by the tree
- *FromUser::Reset()*. Empties the graph and the tree.

3.2.2. Classes description

The vertex class described in precedent chapter was adapted for an HG; nevertheless, in order to create a tree, this class has to be extended. The new class and its inheritance relation are shown in Figure 38. This figure shows that three parameters have been added: *father*, *DistanceToSource* and its current *FanOut*. It has also been added the corresponding procedures to set and get the three new parameters: *getFather()*, *getDistance()*, *getFO()*, *setFather()*, *setDistance()*, *setFO()*. Three more functions have been added to the tree creation algorithm: *decFO()* decrements the current fan-out, *incFO()* increments the current fan-out and *updateDistance(Integer)* updates the current fan-out.

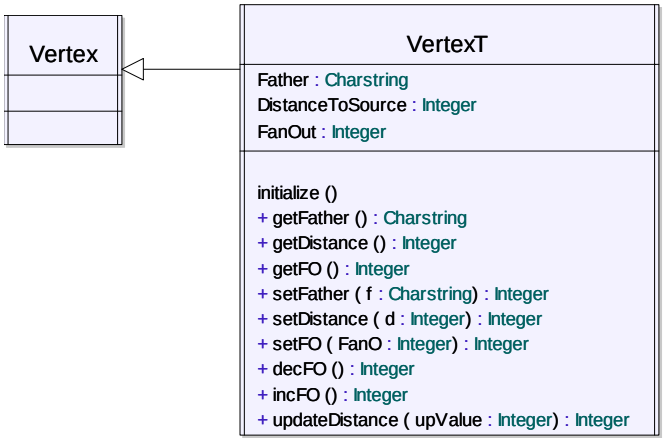

Figure 38: Class VertexT

Figure 24 has been changed in order to contain a list of *VertexT* instead of *Vertex*.

Class *Edge* has also been extended in order to reflect the behavior explained previously. It has been created an enumeration called *Labels* which can take as values: *Normal, Deleted or Indispensable*. It has been added a parameter lab of type *Labels* to class *Edge* and two methods for setting and getting this new parameter: *getLabel()* and *setL()*. Method *setEdge()* has also been changed in order to integrate this new parameter. New class *Edge* and enumeration *Labels* are shown in Figure 39.

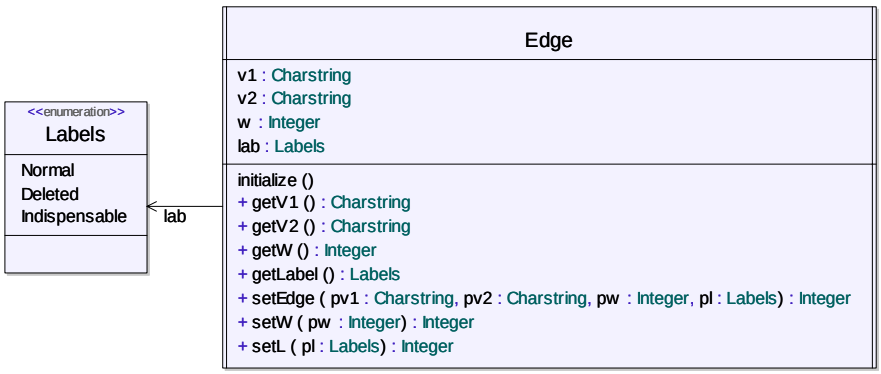

Figure 39: New class Edge and enumeration Labels

Figure 40 shows the relations among all necessary classes needed to construct a Degree Bounded Shortest Path Tree. This figure shows that class *HGraph* contains a list of *VertexT* instead of *Vertex*.

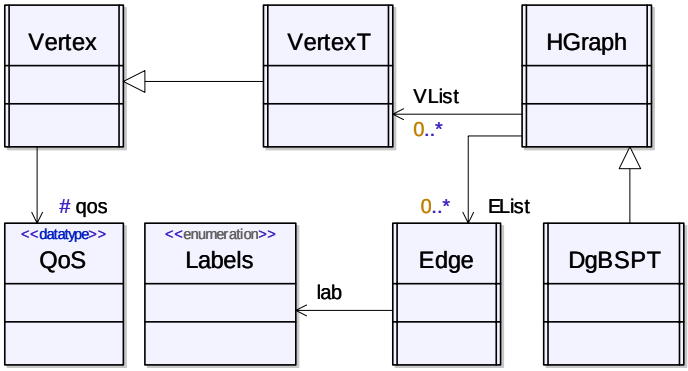

Figure 40: Simplified class diagram

Finally, it has been created a class representing the DgB-SPT algorithm. This class and its behavior will be explained in next section.

3.2.3. Algorithm behavior model

Class *DgBSPT* implements the algorithms explained previously. This class inherits all attributes and methods from *Hgraph* and adds an attribute *FO* which represents the maximal output degree limit within the tree.

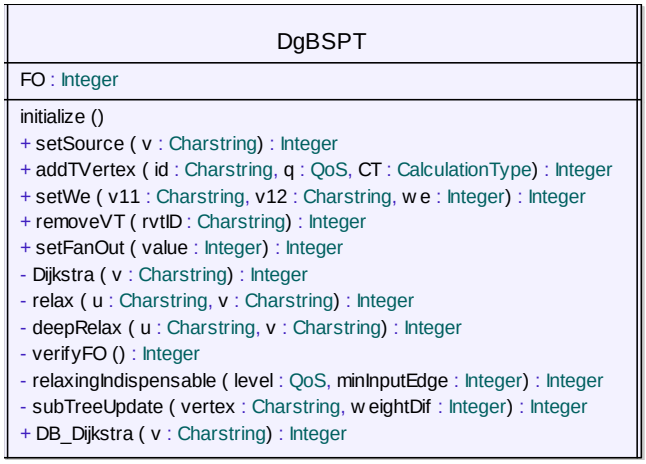

Figure 41: Class DgB-SPT

The most important methods implemented within this class are:

- initialize(). This operation initializes the inherited attributes by calling the *initialize()* operation inherited from *HGraph* and then goes to state *IdleT*.
- *setFanOut(Integer)*. States the maximal fan-out limit to be fulfilled by vertices.
- *setSource(Charstring)*. Searches for the given vertex and defines it as the source, i.e. set its parent as “none” and its distance to the source as 0 (Figure 42).

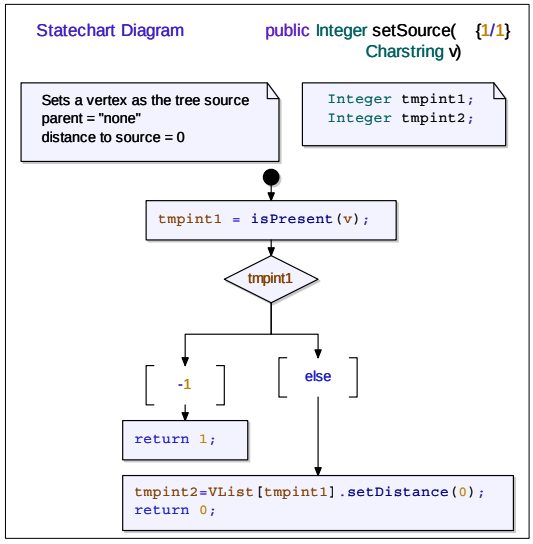

Figure 42: setSource() method description

- *setWe(Charstring, Charstring, Integer)*. Sets the weight of a given edge.

- *verifyFO()*. This function verifies whether the vertices violate the FO value or not. If so, then the location of the more overloaded vertex is returned. (Figure 43).

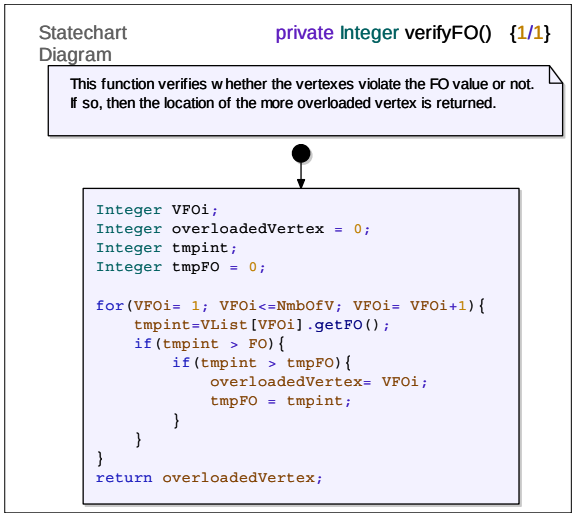

Figure 43: verifyFO() method

- *relax(Charstring, Charstring)*. This function implements the relaxation procedure explained previously. If trying to relax a non positive edge, then a -1 value is returned. If trying to relax non-Normal Edge (i.e. deleted or indispensable) then a -1 value is returned

- *relaxingIndispensable(QoS, Integer)*. When adding a new vertex, it is possible that the graph already contains deleted and indispensable edges. An edge is set *Indispensable* when all other edges arriving to the same level are set to *Deleted*, and so, it is the only edge to keep the level connected. All *Indispensable* edges arriving to

the new vertex level should be changed into *Normal* because it is not longer the only edge to keep the level connected. In the same way, all indispensable edges going to lower levels should be changed into *Normal* ones. This method sets to *Normal* all *Indispensable* edges arriving to the new vertex level and all *Indispensable* edges going to lower levels

- *deepRelax(Charstring, Charstring)*. Similar to relax function, but if an edge (*u,v*) is relaxed, then *v*'s subtree is updated

- *Dijkstra(Charstring)*. This is a dynamic version of Dijkstra's algorithm. This operation will relax all input neighbors of a new vertex and then will relax all output neighbors. After that, as explained before, the new vertex is considered to be integrated in the tree and knows its parent and distance to the source. It is necessary to note that this method considers that the vertex to be integrated in the tree has already been added to the graph and that all needed edges have been also added

- *addTVertex(Charstring, QoS, CalculationType)*. This method adds a new vertex to a graph and to the tree (Figure 44)

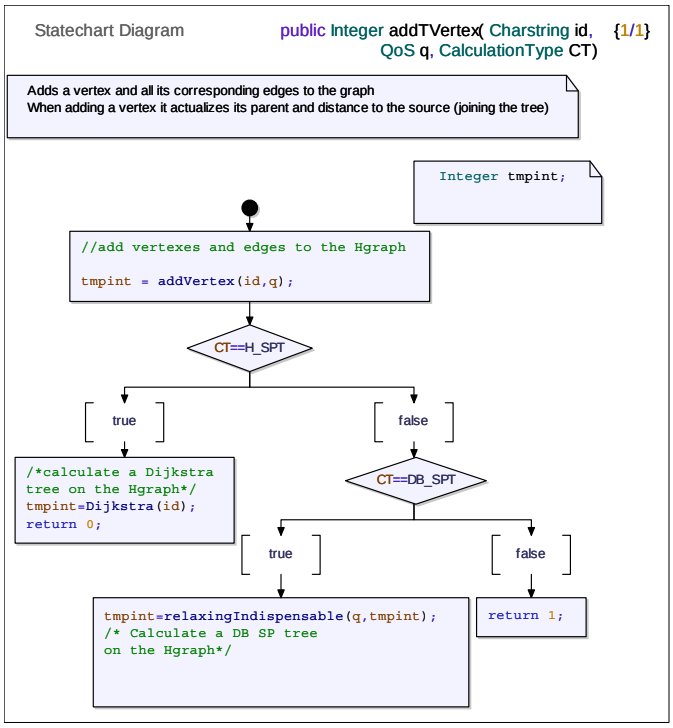

Figure 44: addTVertex() diagram

- *DB_Dijkstra(Charstring)*. This is the most important and the longer method in the algorithm model. It uses four auxiliary functions which are explained after the main function. *DB_Dijkstra* relaxes all *input* edges of a new vertex and then relaxes all *output* edges. After that, the new vertex is considered to be integrated within the tree. Then, the *fan-out constraints* verification and correction is performed. First, the operation verifies if FO constraint is violated; if so it selects an edge to be pruned. This procedure is repeated until the FO constraint is fulfilled by all nodes in the tree

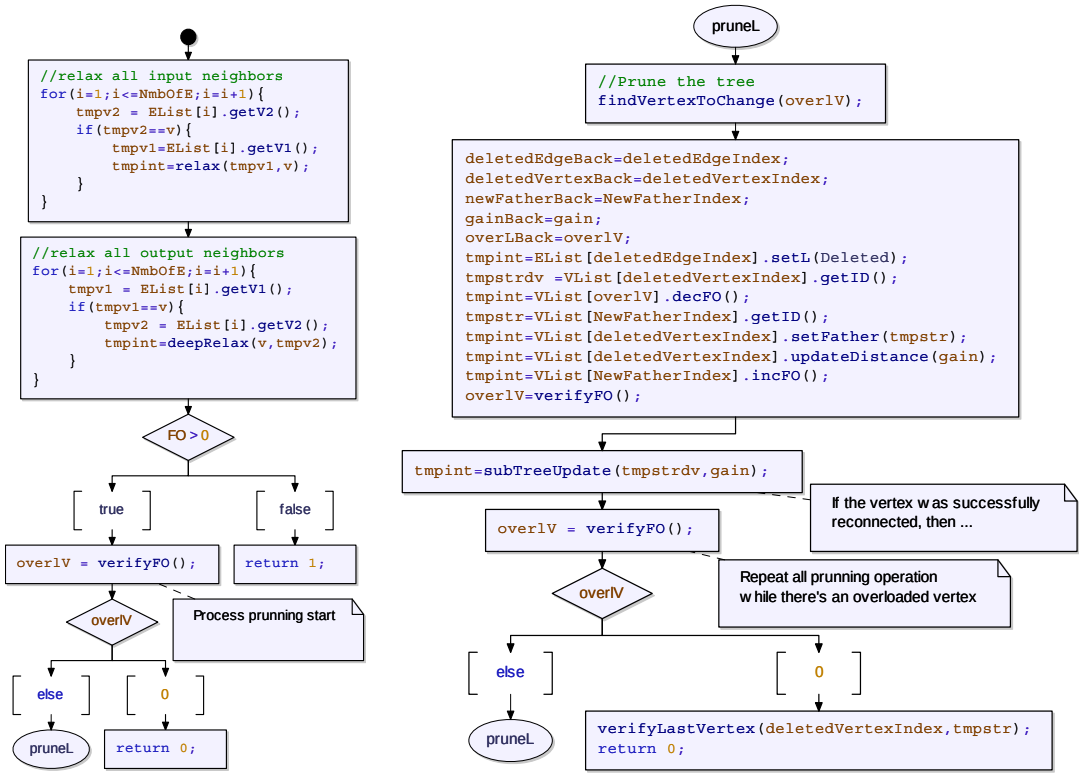

Figure 45: function DB_Dijkstra

- *DB_Dijstra::isInArray(Charstring)*. Verifies if vertex *vIA* is in *oneLevelSubTree*; if so it returns true, otherwise it returns false

```
Integer gain;
Integer deletedVertexIndex;
Integer deletedEdgeIndex;
Integer NewFatherIndex;
Charstring [0..*] oneLevelSubTree;

private Boolean isInArray( Charstring vIA) {
    Integer k;
    Boolean is=false;

    for(k=1;k<=oneLevelSubTree.length();k=k+1){
        if(oneLevelSubTree[k]==vIA){
            is = true;
        }
    }
    return is;
}
```

Figure 46: isInArray method

- *DB_Dijstra::verifylastVertex(Integer vfLV,Charstring vfLVNFN)*. Verifies if vertex *vfLV* is the last vertex within its QoS level. If so, it verifies the number of *Normal* edges arriving to it. If there is only one edge arriving to it, then the operation sets edge's label to *Indispensable*

```
private void verifyLastVertex(Integer vfLV,Charstring vfLVNFN){
    Integer vfi;
    Integer vfj=0;
    Integer usedEdgeIndex;
    QoS vfqos;
    Boolean cont = true;
    Edge vfE;
    Charstring vfLVName;

    vfLVName=VList[vfLV].getID();

 //is it the last vertex in its level?
    vfqos=VList[vfLV].getQoS();
    for(vfi=1;vfi<=NmbOfV;vfi=vfi+1){
        if(vfi!=vfLV)
            if(VList[vfi].getQoS()==vfqos)
                cont=false;
    }

    //if it is the las vertex in tis level
    if(cont){
        for(vfi=1;vfi<=NmbOfE;vfi=vfi+1){
            vfE=EList[vfi];
            if(vfE.getV2()==vfLVName){
                if(vfE.getLabel()==Normal){
                    vfj=vfj+1;
                    if(vfE.getV1()==vfLVNFN)
                        usedEdgeIndex=vfi;
                }
            }
        }
    }

    //if there is an only Normal edge to the vertex
    if(vfj==1)
        vfi=EList[usedEdgeIndex].setL(Indispensable);
}
```

Figure 47: verifyLastVertex method

- *DB_Dijstra::findOneLevelSubtree(Integer vfo)*. Finds all *vfo* children within the same level

```
private void findOneLevelSubTree(Integer vfo){
    Integer ii;
    Integer jj;
    QoS QoSRef;
    Charstring vertexName;
    VertexT vj;


    //Empty the array
    while(oneLevelSubTree.length()>0)
        oneLevelSubTree.remove(1);

    //find the "Vfo"'s subtree within the same level
    vertexName = VList[vfo].getID();
    oneLevelSubTree.append(vertexName);
    QoSRef = VList[vfo].getQoS();

    for(ii=1;ii<=oneLevelSubTree.length();ii=ii+1){
        for(jj = 1; jj <= NmbOfV; jj = jj + 1) {
            vj = VList[jj];
            if (vj.getQoS() == QoSRef)
                {
                    if (oneLevelSubTree[ii]==vj.getFather())
                        {
                            oneLevelSubTree.append(vj.getID());
                        }
                }
        }
    }
}
```

Figure 48: findOneLevelSubTree method

  - *DB_Dijstra::findVertexToChange(Integer)*. Implements Algorithm 8 in order to select an edge to be pruned within the tree.

- *subTreeUpdate(Charstring, Integer)*. When pruning an edge, it is necessary to reconnect the disconnected vertex. By doing so, the vertex increments its distance-to-the-source because it has lost its shortest-path. This operation is called when a pruned vertex is reconnected. This operation may update the distance of the reconnected sub-tree. It will be so necessary to search for a better route to the source (because their distance-to-the-source has incremented) for the whole updated sub-tree

- *removeVT(Charstring)*. This operation removes a vertex and all its related edges from the graph and the tree. This procedure uses the insertion process and is explained in previous subsection.

3.2.4. Algorithm Validation

As said before, the CASE tool called TAU G2 by Telelogic has been used in order to define and validate the algorithm model. This subsection presents the algorithm validation and the automatically generated sequence diagrams.

The algorithm first goes to the inherited first state of *HGraph* and then to its first state (Figure 49).

The second step in the algorithm utilization is the fan-out definition. In this simulation, maximal fan-out limit was stated to 1 (Figure 49).

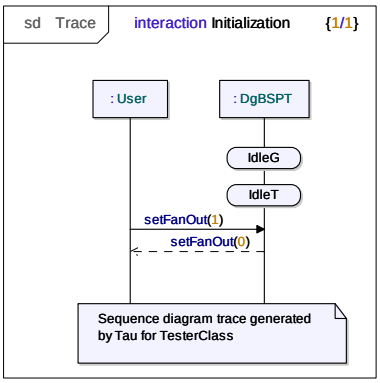

Figure 49: Algorithm initialization and fan-out setting

Then, the first vertex can be added. In this case, the new vertex is called *Sender* and it asks for the maximal QoS, i.e. QoS_0. The third parameter in the message from *user* to *algorithm* indicates that the vertex will be added by using the *DgB-SPT* method. After receiving the *addVertex* message, the algorithm creates the indicated vertex and looks for any vertex exceeding the maximal fan-out limit. As this is the first vertex added, the process finishes with no more actions (Figure 50).

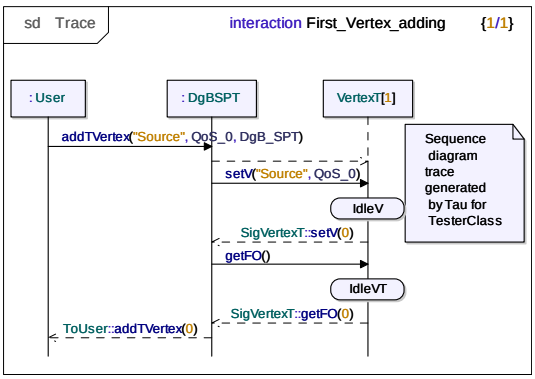

Figure 50: first vertex adding

Now the user sets this first vertex as source. The algorithm sets its distance to the source as 0 and its parent as *none*.

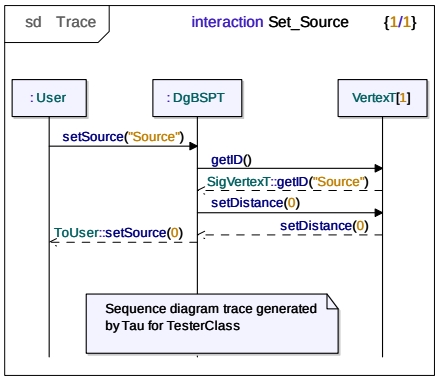

Figure 51: Set Source procedure

The user decides to add a new vertex ED_1 with QoS_1 by using the DgB_SPT method. The algorithm verifies that this vertex does not exist and creates and sets the new vertex. Then, it looks for edges having a QoS higher than or equal to the new one and creates the corresponding edges. In this case, the algorithm creates an edge from vertex Source to vertex ED_1 and asks the user for news edge's weight (Figure 52).

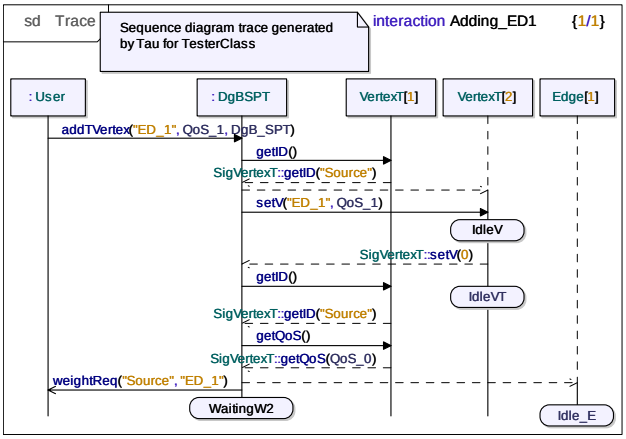

Figure 52: adding vertex ED_1 and its edge's weight and asking for new edge's weight

The user sends the weight back, in this case 30; then the algorithm sets the edge's weight. Then the *relaxIndispensable* method explained previously is called.

The DB_Dijkstra method is then run. This method calls first the *relax* method on all new vertex's input edges (Figure 53).

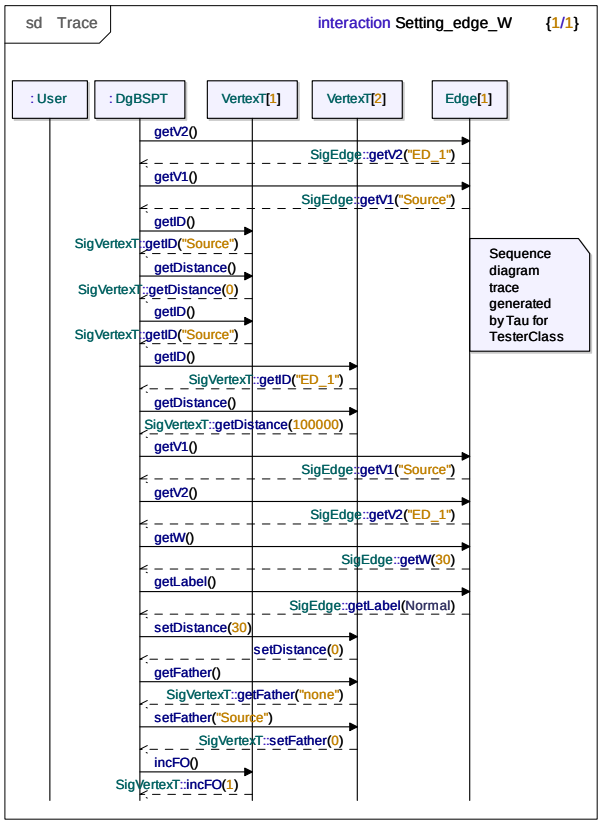

Figure 53: relaxing all ED_1's input/output edges

Finally, the *deepRelax* method is applied on all new vertex output edges. In this case, there are no output edges for the new vertex; then the algorithm verifies if all the vertices fulfill the given maximal output limit.

The following figures show the process of adding another vertex. In Figure 54a the user sends the *addVertexT* message, the algorithm verifies that this new vertex does not exist, and creates and sets it as ED_2 with QoS_2. Next, in Figure 54b, the algorithm creates the edge (*source, ED_2*) and asks the user for its weight. Then, the user answers the request and the algorithm sets the edge's weigh as 50.

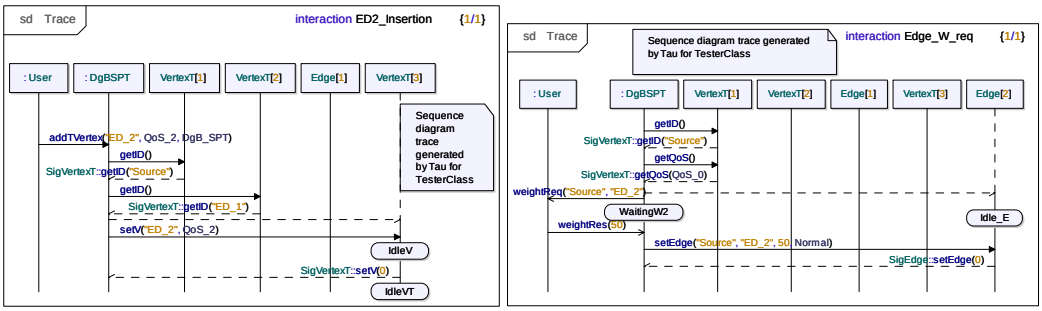

Figure 54: a) new vertex insertion; b) new edge's weight request and set

The algorithm then creates the edge (*ED_1, ED_2*) and asks the user for its weight. Then, the user answers the request and the algorithm sets the edge's weigh as 50. Next, in the algorithm applies the *relaxIndispensable* process.

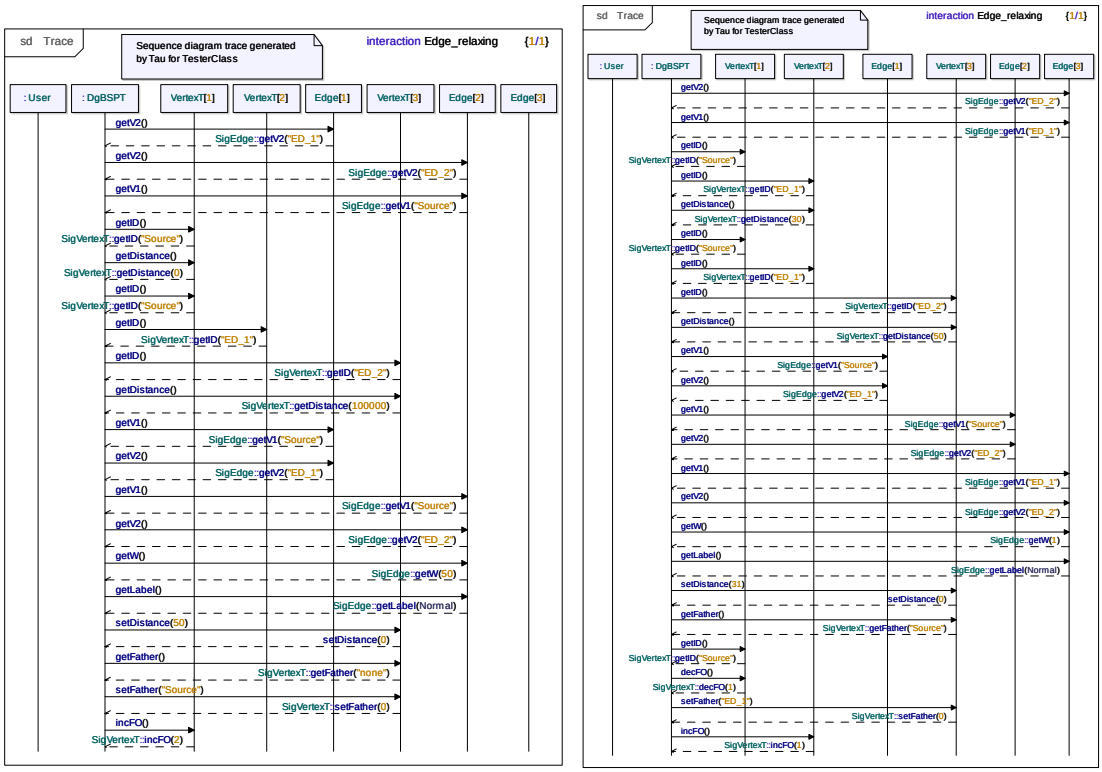

Figure 55: relaxing edge (source, ED_2) and edge (ED_1 , ED_2)

Figure 55 shows the beginning of the *DB_Dijkstra* process; the algorithm has to relax all ED_2's input edges. As shown before, the first added edge was (*source, ED_2*), so this is the first relaxed edge. The algorithm selects this edge and sets the source as ED_2's father and sets its distance to 50. Then, the algorithm increments the *source* FO.

In Figure 55 the algorithm searches for other input edges and relaxes edge (*ED_1, ED_2*). Then it decides to set ED_1 as ED_2's father and its distance to the source as 31. Next, the algorithm decrements *source's* FO and increments *ED_1's* FO.

Finally, the algorithm searches for possible ED_2's output edges and verifies if either global FO limit is fulfilled or not (Figure 56).

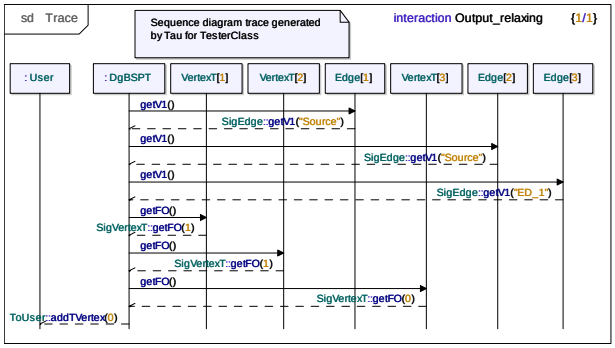

Figure 56: relaxing all ED_2's output edges and verifying the FO limit

The resulting graph and tree are given in Figure 57.

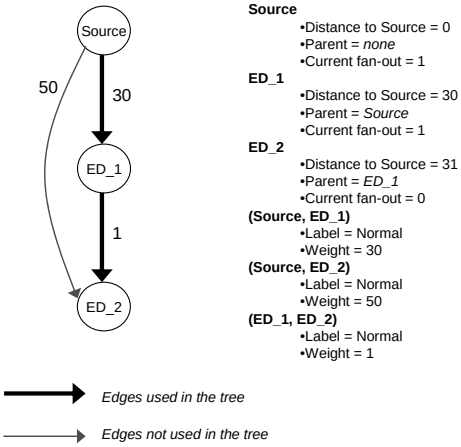

Figure 57: graph and tree used for algorithm validation

Many simulations have been conducted and these simulations have permitted to validate the correct behavior of the model. As said before, these models and simulation were performed with the aid of a CASE tool called TAU G2 created by Telelogic [TAUG2]. This tool allows the analysis of the model by creating a *coverage statistics table*. This table contains, for each operation defined in the system, the number of statements and transitions that were covered. It provides also a per-operation covered percentage. Table 2 shows the analysis of the model presented previously. This table shows that most of the operations have been completely covered. However, in the performed simulations, there are some non-used operations as *getNmbOfE*, *getNmbOfV*, *indexOf*, *getVertexAt*, *setWe* or *Dijkstra*. These operations are accessory methods in the API and can be used by programmers. Nevertheless, in this simulation they do not change the global model behavior. Taking these non-used operations off, the model has been covered in 92.62%.

Operation	Path	Kind	Nmb	Covered	%Covered
Vertex	::DBSPT8::Vertex	Statements	6	4	66
Vertex	::DBSPT8::Vertex	Transitions	6	2	33
getID	::DBSPT8::Vertex::getID	Statements	2	2	100
getID	::DBSPT8::Vertex::getID	Transitions	1	1	100
getQoS	::DBSPT8::Vertex::getQoS	Statements	2	2	100
getQoS	::DBSPT8::Vertex::getQoS	Transitions	1	1	100
setV	::DBSPT8::Vertex::setV	Statements	4	4	100
setV	::DBSPT8::Vertex::setV	Transitions	1	1	100
VertexT	::DBSPT8::VertexT	Statements	7	5	71
VertexT	::DBSPT8::VertexT	Transitions	20	10	50
getFather	::DBSPT8::VertexT::getFather	Statements	2	2	100
getFather	::DBSPT8::VertexT::getFather	Transitions	1	1	100
getDistance	::DBSPT8::VertexT::getDistance	Statements	2	2	100
getDistance	::DBSPT8::VertexT::getDistance	Transitions	1	1	100
getFO	::DBSPT8::VertexT::getFO	Statements	2	2	100
getFO	::DBSPT8::VertexT::getFO	Transitions	1	1	100
decFO	::DBSPT8::VertexT::decFO	Statements	3	3	100
decFO	::DBSPT8::VertexT::decFO	Transitions	1	1	100
incFO	::DBSPT8::VertexT::incFO	Statements	3	3	100
incFO	::DBSPT8::VertexT::incFO	Transitions	1	1	100
setFather	::DBSPT8::VertexT::setFather	Statements	3	3	100
setFather	::DBSPT8::VertexT::setFather	Transitions	1	1	100
setDistance	::DBSPT8::VertexT::setDistance	Statements	3	3	100
setDistance	::DBSPT8::VertexT::setDistance	Transitions	1	1	100
setFO	::DBSPT8::VertexT::setFO	Statements	3	3	100
setFO	::DBSPT8::VertexT::setFO	Transitions	1	1	100
updateDistance	::DBSPT8::VertexT::updateDistance	Statements	3	3	100
updateDistance	::DBSPT8::VertexT::updateDistance	Transitions	1	1	100
Edge	::DBSPT8::Edge	Statements	6	6	100
Edge	::DBSPT8::Edge	Transitions	8	7	87
getV1	::DBSPT8::Edge::getV1	Statements	2	2	100
getV1	::DBSPT8::Edge::getV1	Transitions	1	1	100
getV2	::DBSPT8::Edge::getV2	Statements	2	2	100
getV2	::DBSPT8::Edge::getV2	Transitions	1	1	100
getW	::DBSPT8::Edge::getW	Statements	2	2	100
getW	::DBSPT8::Edge::getW	Transitions	1	1	100
getLabel	::DBSPT8::Edge::getLabel	Statements	2	2	100
getLabel	::DBSPT8::Edge::getLabel	Transitions	1	1	100
setEdge	::DBSPT8::Edge::setEdge	Statements	6	6	100
setEdge	::DBSPT8::Edge::setEdge	Transitions	1	1	100
setL	::DBSPT8::Edge::setL	Statements	3	3	100
setL	::DBSPT8::Edge::setL	Transitions	1	1	100
HGraph	::DBSPT8::HGraph	Statements	10	10	100
HGraph	::DBSPT8::HGraph	Transitions	8	1	12
addVertex	::DBSPT8::HGraph::addVertex	Transitions	10	2	20
getNmbOfE	::DBSPT8::HGraph::getNmbOfE	Statements	2	0	0
getNmbOfE	::DBSPT8::HGraph::getNmbOfE	Transitions	1	0	0
getNmbOfV	::DBSPT8::HGraph::getNmbOfV	Statements	2	0	0
getNmbOfV	::DBSPT8::HGraph::getNmbOfV	Transitions	1	0	0
remove	::DBSPT8::HGraph::remove	Statements	36	35	97
remove	::DBSPT8::HGraph::remove	Transitions	1	1	100
isPresent	::DBSPT8::HGraph::isPresent	Statements	11	11	100
isPresent	::DBSPT8::HGraph::isPresent	Transitions	1	1	100
findEdge	::DBSPT8::HGraph::findEdge	Statements	11	9	81
findEdge	::DBSPT8::HGraph::findEdge	Transitions	1	1	100
DgBSPT	::DBSPT8::DgBSPT	Statements	2	2	100
DgBSPT	::DBSPT8::DgBSPT	Transitions	15	5	33
verifyFO	::DBSPT8::DgBSPT::verifyFO	Statements	14	14	100
verifyFO	::DBSPT8::DgBSPT::verifyFO	Transitions	1	1	100
setSource	::DBSPT8::DgBSPT::setSource	Statements	6	5	83
setSource	::DBSPT8::DgBSPT::setSource	Transitions	1	1	100
addTVertex	::DBSPT8::DgBSPT::addTVertex	Statements	10	7	70
addTVertex	::DBSPT8::DgBSPT::addTVertex	Transitions	1	1	100
setWe	::DBSPT8::DgBSPT::setWe	Statements	6	0	0
setWe	::DBSPT8::DgBSPT::setWe	Transitions	1	0	0
removeVT	::DBSPT8::DgBSPT::removeVT	Statements	84	74	88
removeVT	::DBSPT8::DgBSPT::removeVT	Transitions	1	1	100
@Newrec	::DBSPT8::DgBSPT::removeVT::@Newrec	Statements	5	5	100
setFanOut	::DBSPT8::DgBSPT::setFanOut	Statements	3	3	100
setFanOut	::DBSPT8::DgBSPT::setFanOut	Transitions	1	1	100
Dijkstra	::DBSPT8::DgBSPT::Dijkstra	Statements	20	0	0
Dijkstra	::DBSPT8::DgBSPT::Dijkstra	Transitions	1	0	0
relax	::DBSPT8::DgBSPT::relax	Statements	22	21	95
relax	::DBSPT8::DgBSPT::relax	Transitions	1	1	100
deepRelax	::DBSPT8::DgBSPT::deepRelax	Statements	51	36	70
deepRelax	::DBSPT8::DgBSPT::deepRelax	Transitions	1	1	100
relaxingIndispensable	::DBSPT8::DgBSPT::relaxingIndispensable	Statements	26	25	96

relaxingIndispensable	::DBSPT8::DgBSPT::relaxingIndispensable	Transitions	1	1	100
subTreeUpdate	::DBSPT8::DgBSPT::subTreeUpdate	Statements	29	27	93
subTreeUpdate	::DBSPT8::DgBSPT::subTreeUpdate	Transitions	1	1	100
DB_Dijkstra	::DBSPT8::DgBSPT::DB_Dijkstra	Statements	46	44	95
DB_Dijkstra	::DBSPT8::DgBSPT::DB_Dijkstra	Transitions	1	1	100
findVertexToChange	::DBSPT8::DgBSPT::DB_Dijkstra::findVertexToChange	Statements	63	60	95
findVertexToChange	::DBSPT8::DgBSPT::DB_Dijkstra::findVertexToChange	Transitions	1	1	100
isInArray	::DBSPT8::DgBSPT::DB_Dijkstra::isInArray	Statements	10	10	100
isInArray	::DBSPT8::DgBSPT::DB_Dijkstra::isInArray	Transitions	1	1	100
findOneLevelSubTree	::DBSPT8::DgBSPT::DB_Dijkstra::findOneLevelSubTree	Statements	23	22	95
findOneLevelSubTree	::DBSPT8::DgBSPT::DB_Dijkstra::findOneLevelSubTree	Transitions	1	1	100
verifyLastVertex	::DBSPT8::DgBSPT::DB_Dijkstra::verifyLastVertex	Statements	28	26	92
verifyLastVertex	::DBSPT8::DgBSPT::DB_Dijkstra::verifyLastVertex	Transitions	1	1	100
lowerThan	::DBSPT8::lowerThan	Statements	18	12	66

Table 2: statements and transition coverage statistics for the algorithm model

3.3. Simulations and Outcomes

The validation process permits to validate the correctness, but it does not give any measure of the performance. In order to measure the performance with respect to the original Dijkstra's algorithm, the hierarchized graph and the DgB-SPT algorithm have been implemented in JAVA. Then, a set of tests have been performed. These tests correspond to those explained in section 3.1.2. Let us recall these tests.

Network under test has the following characteristics:

- The network contains 300 nodes (plus the source)
- 8 QoS levels were defined and were distributed to the vertices by using a uniform function.

It has been created an HG from this network. Edge weights within the HG were randomly assigned from 10 to 200 by using a uniform function. After HG creation, Dijkstra's algorithm was used in order to create a SPT which spans all the vertices from the source. This test was repeated 100 times. On each test it was measured the:

- Average distance from all vertices to the source
- Average fan-out of the vertices
- Maximal fan-out of the tree
- Maximal distance to the source.

In order to compare the outcomes obtained from DgB-SPT simulations, let us recall the results obtained when applying the Dijkstra's algorithm on a hierarchized graph.

Concerning the fan-out, Figure 29 showed that most of vertices have a small fan-out: 75% of vertices have FO=0 and 90% have a fan-out lower than or equal to 2. Nevertheless, at the other extreme, 0.6% of vertices have a fan-out of at least 20 and 2.4% have a fan-out of at least 10.

Concerning the source, it has always the maximal fan-out that can grow until 59 (see Figure 28) and it is never lower than 32.

Concerning the distance to the source, Figure 30 shows that, even if edge weights were uniformly distributed from 10 to 200, the mean distance to the source is considerably low (27.962) while the maximal distance is 90. Figure 31 showed that 90% of vertices have a distance to source lower than or equal to 40 and 98.4% have a distance lower than or equal to 50.

Let us now expose the results obtained when constraining the value of the maximal fan-out to 20, 12, and 4 and let us compare these results with those obtained when no fan-out constraint was given.

The distribution curve concerning the fan-out is shown in Figure 58. The percentages of vertices having fan-out equal to 0 are 73%, 72% and 61% when the fan-out limit is constrained to 20, 12 and 4 respectively. Figure 59 shows that the FO distribution curves for maximal fan-out limits equal to 20 and 12 are very similar to the one observed when no fan-out limitation was defined.

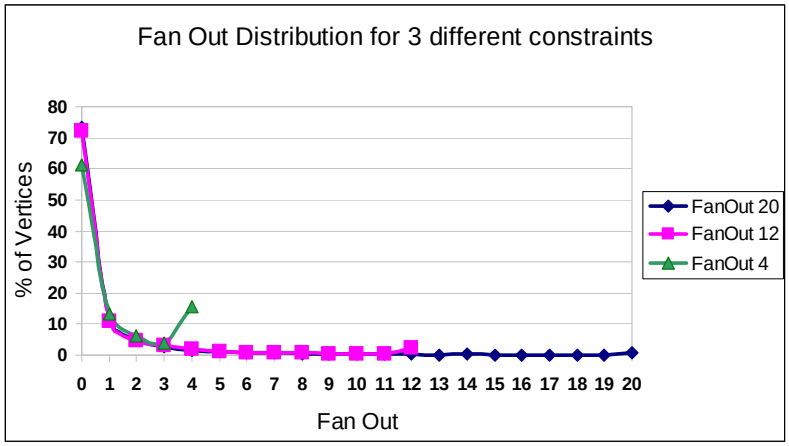

Figure 58: Fan-Out Distribution for 3 different constraints

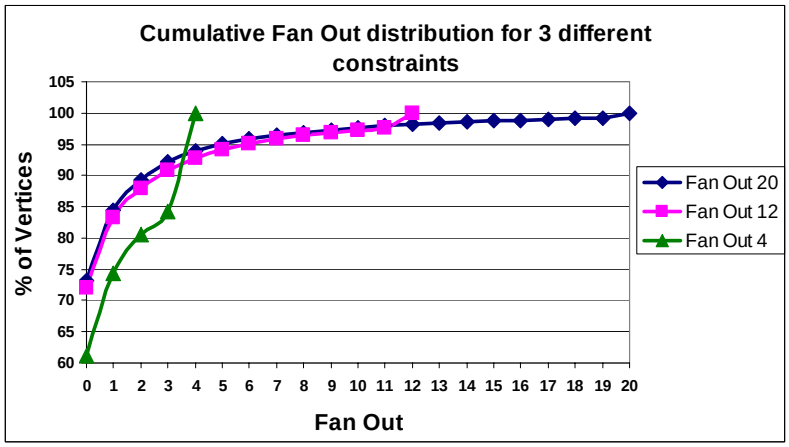

Figure 59: Cumulative Fan Out distribution for 3 different constraints

So we observe that:

- When no maximal fan-out limit is defined, 75% of vertices have fan-out=0
- When maximal fan-out limit is set to 20 and 12, 73% 72% of vertices have fan-out=0 respectively.

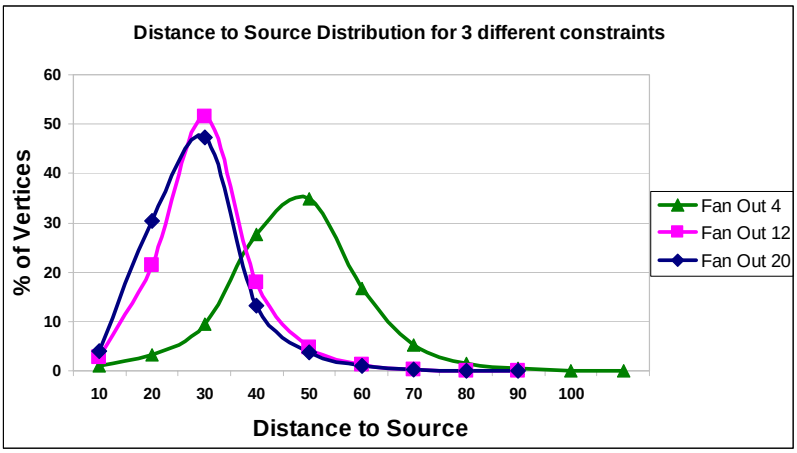

Figure 60: Distance to Source Distribution for 3 different constraints

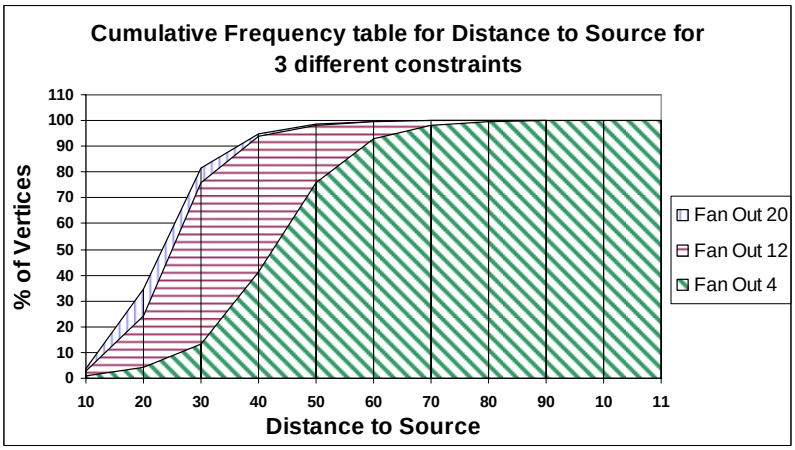

Figure 61: Cumulative Frequency table for Distance to Source for 3 different fan-out constraints

As expected, the average distance to source changes when constraining the fan-out. When no maximal fan-out limit is defined, the mean distance to source is 27.9, while it is 28.7, 30.5 and 47.42 when fan-out limit is fixed to 20, 12 and 4 respectively. Again, limiting the fan-out to 20 or 12 does not change considerably the tree performance. However, this mean increase does not affect all the vertices; as shown in Figure 31 and Figure 61, 90% of the vertices have a distance to the source lower than or equal to 40 when limiting the fan-out to infinite, 20 and 12, and it goes until 60 when limiting the fan-out to 4

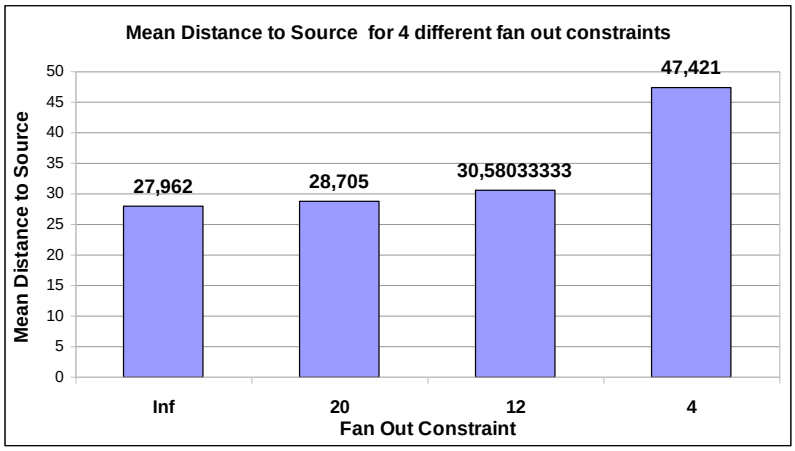

Figure 62: Distance to Source mean for 4 different fan-out constraints

Table 3 summarizes these results.

Measure \ FO limit	No Fan-Out limit	Fan-Out limit=20	Fan-Out limit=12	Fan-Out limit=4
Mean distance to source	27.962	28.705	30.58	47.421
Maximal distance-to-source	90	90	90	110
Distance-to-source for 75% of vertices	≤ 30	≤ 30	≤ 30	≤ 50
Distance to source for 80 % of vertices	≤ 30	≤ 30	≤ 40	≤ 60
Distance to source for 90 % of vertices	≤ 40	≤ 40	≤ 40	≤ 60
Distance to source for 95 % of vertices	≤ 50	≤ 50	≤ 50	≤ 70
Distance to source for 99 % of vertices	≤ 60	≤ 60	≤ 60	≤ 80
Maximal fan-out	59	20	12	4
Vertices having fan-out=0	75.09 %	73.24 %	72.07 %	61.22 %
Vertices having fan-out ≥ 10	2.42%	2.67%	3.10%	0%
Vertices having fan-out ≥ 20	0.62%	0.78%	0%	0%

Table 3: Outcomes summary

We can conclude from this that it is possible to find a compromise between the distance to the source and the fan-out limitation in order to fulfill session characteristics. For example, if the fan-out is fixed to 8, then the average distance to the source is 33.9 (27.8 when FO=infinite) and 89.7% of vertices have distance lower than or equal to 40 (94.5% when FO=infinite).

3.4. Chapter summary and discussion

This chapter has presented an analysis on the need of multicast algorithms taking into account the users' QoS constraints (particularly for a user QoS oriented multimedia multicast service).

Because of the lack of user oriented QoS multicast trees and algorithms, this chapter proposes a new algorithm called Degree Bounded Shortest Path Tree. In a first part, this algorithm has been presented as a pseudo-code and its shortest-path have been demonstrated

based on Dijkstra's algorithm properties. Then, the algorithm has been modeled by using MDA and UML; a set of test targeting the algorithm validation have been presented. Finally, the algorithm has been implemented in JAVA and a set of tests have been performed on it and the results and measurements presented. All the UML diagrams and simulations are available at author's web site [GAR-W].

In particular, it has been shown that the DgB-SPT algorithm can be defined and tuned in order to give either more weight to shortest-paths (SPT) or to degree-bounding (DBT). It has also been shown that setting the maximal fan-out between 12 and 8 can give a spanning tree performing very similarly to an SPT while solving its overloading problem.

Nevertheless I order to implement and to deploy this algorithm we still need: on one hand a session protocol in order to drive the required QoS information from the users to the server; and on the other hand a deployment method allowing to dynamically load and configure the FPTP receiving/sending proxies in the receiver.

The next chapter proposes a solution to these two problems.

Chapter 4

System Integration

Precedent chapters have stated the basis of a new user's QoS oriented multicast service.

Chapter 2 has explained the needed modifications to FPTP proxies in order to relay many local multicast networks by using single FPTP connections. This new architecture, called M-FPTP, permits to define a different QoS for each proxy in the network. In the same chapter, it has been shown that, in a multimedia multicast session, all nodes are not available to receive the same QoS. It has also been shown that current multicast models do not take into account this user-oriented QoS in order to create a spanning tree relaying the network elements. It is then proposed a new network organization model called Hierarchized Graph which takes into account, at the same time, the users QoS requirements and the network performances.

Chapter 3 has shown that the most adequate type of spanning tree for a MM session is an SPT. It has also shown that the application of standard SPT algorithms, such as Dijkstra's one, on an HG can lead into a overloading problem in the source. In order to solve this problem, it has been proposed a new algorithm called Degree-Bounded Shortest-Path-Tree (DgB-SPT). DgB-SPT creates a degree-bounded-tree which minimizes the distance-to-the-source for all the vertices. This algorithm is based on Dijkstra's one.

In order to integrate all these propositions together, this chapter proposes a new protocol called Simple Session Protocol for QoS Multicast (SSP-QoM). This protocol receives the login requests from MM clients and their QoS requirements and creates an HG; then it dynamically loads a measurement module on involved proxies, recuperates the measures following the HG structure and creates a DgB-SPT; finally, the protocol dynamically deploys the modified FPTP proxies by following the tree structure.

SSP-QoM accepts dynamic users login and out. This protocol is modeled by using UML 2.0 and the TAU G2 CASE tool and is validated by simulating the UML models.

This chapter is organized as follows. Section 4.1 establishes the base network configuration and the protocol expected behavior. Then, section 4.2 exposes the protocol architecture and the expected modules on each network element. After that, section 4.3 exposes the protocol UML model. This model is briefly explained by showing firstly its interfaces and interactions with the

users. Then, for each software element, it has been created a class; these classes and their relationships are explained by using a class diagram. After that, their interactions and interchanged messages and parameters are explained by showing two UML architecture diagrams. Finally, the model is validated in section 4.4 by simulating the UML model. As the protocol design is based on MDA and UML, the chapter is supported by a set of UML diagrams and concepts.

4.1. Protocol requirements specification

Nowadays there exist many session protocols which are adapted to different sets of requirements or attributes optimization. Some of them are adapted to point-to-point transmissions or to multicast transmissions. Some others are adapted to multimedia flows or to reliable flows. Others can combine more than one property, for example multimedia multicast sessions, reliable multicast sessions, etc. Nevertheless, to our current knowledge, none of them are adapted to a user's QoS multicast.

This section establishes the requirements of a new protocol called Simple Session Protocol for Quality of Service Multicast (SSP-QoM). This protocol receives the user's login/logout requests and their QoS requirements; it creates an HG and collects the distances between the proxies and builds a DgB-SPT; then it deploys the M-FPTP modules on the participating proxies by following the tree structure. Finally, SSP-QoM uses a programmable network technology in order to dynamically upload the needed modules.

4.1.1. Network configuration

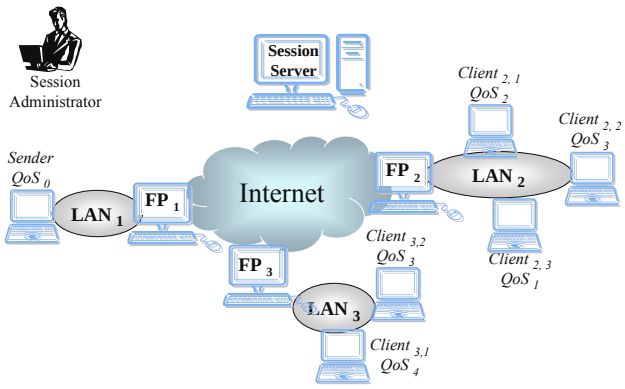

Figure 63: Network configuration used for examples

The network configuration used for our work is formed by a set of LANs with local multicast, interconnected by the internet without a defined quality of service. The clients are placed within the LANs. The clients will access the session flows by using some access points called FPTP Proxies. The multicast tree will be created among the proxies through the internet by using single FPTP connections for interconnecting the local area networks. The session administration and control and the tree creation will be done by a server called "*session server*". The network infrastructure shown in Figure 63 will be used as example in order to explain the protocol behavior.

In order to better explain the expected behavior, some naming conventions are given:

- Let us give an ID FP_x for each FPTP Proxy and ID FP_1 for the FPTP Proxy serving the MM source.
- Let us give an ID $Client_{x,n}$ to the client number *n* served by FP_x.

4.1.2. Protocol behavior

4.1.2.1. *Elements tasks*

The tasks assigned to each element in the network (Figure 63) are:

Session Administrator: in the general case it creates and defines the session. The session is defined by some parameters as Session name, Fan-out limit, Members list, Sender ID. The Administrator also controls the session, i.e. decides to open, to start, to stop or to end the session.

Session Server: its function is to manage the session, receive the clients' requests and sending control messages to clients and proxies.

This server creates and modifies the HG and the DgB-SPT. Concerning the clients, it will:

- Receive the clients' login/out requests
- Receive the clients' QoS constraints
- Receive the client's Proxy ID
- Acknowledge clients' requests
- Send *stop*, *wait* and *continue* control messages to clients.

Concerning the proxies, the session server will:

- Make the proxies to dynamically load and unload *Measurement* and *M-FPTP* modules
- Send *Connect*, *Disconnect* and *ChangeQoS* messages to M-FPTP modules
- Send *Measure* message to *Measurement* module
- Receive the distance measures taken by the FPTP proxies.

Sender: it will send data to the session

4.1.2.2. *Session definition and first clients login*

The operations described in this section are illustrated in Figure 64.

1. The session administrator defines the session. In this example, the only mandatory items are the sender name and the maximal fan-out limit. Nevertheless, in a possible extension to the protocol, it would be possible to add some items as users list, session name, security

parameters, etc. In order to show the protocol behavior in a small graph, the fan-out limit is defined as 1

2. Some clients (including the server) send a login session message to the server prior the session starting. The mandatory items in a user login are: its ID or its role within the session (Sender or receiver, for example), its serving proxy ID and its QoS parameters. It is necessary to remark that, as the DgB-SPT is a single source oriented tree, it cannot be created if the sender is not logged within the session. So, the sender has to log into the session before session starting. In this example, *Client* $_{3,1}$ and *Client* $_{3,2}$ are located within the same LAN, *Client* $_{3,1}$ requires a QoS_4 and *Client* $_{3,2}$ requires a QoS_3. As said before, all the clients within the same LAN are receiving the same dataflow through the same local IP multicast address. It has also been said that a *FP* will receive an only QoS. As *Client* $_{3,1}$ and *Client* $_{3,2}$ are requiring different QoS, the *Session Server* has to assign only one QoS to the *FP* and the two *clients*. This can be done by using two criteria: first, the server assigns to the *FP* (and all the clients in the same LAN) the maximal QoS among all the clients. This alternative permits to satisfy all the clients, even those requiring a high QoS. The second alternative is to assign to the *FP* (and all the clients in the same LAN) the minimal QoS among all the clients. This criterion permits to avoid overloading those clients asking for a low QoS. The choice between these two criteria can be a parameter in the session definition. For the moment, the first criterion is taken. Figure 64b shows the QoS assignment for the *FPs* within the HG

3. The server acknowledges the received messages and sends a wait message to the clients

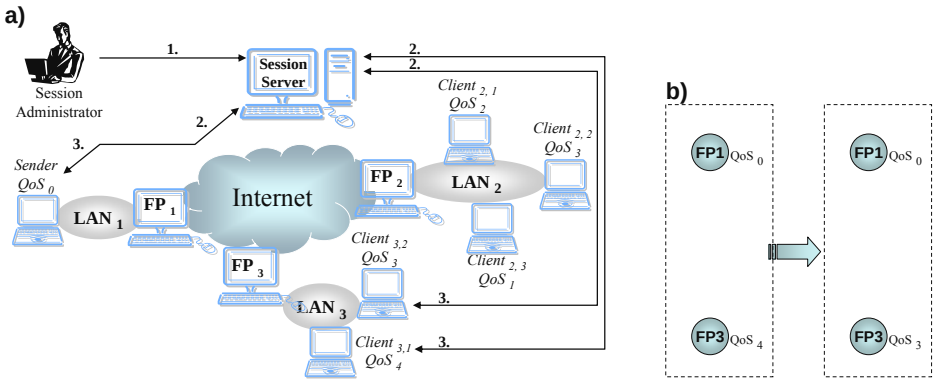

Figure 64: a) Session definition, first login and session start; b) graph structure

4.1.2.3. *Session starting*

The operations described in this section are illustrated in Figure 65 and Figure 66.

4. The *Administrator* starts the session
5. The *Session Server* creates the graph by following the algorithms explained in chapter 2. This algorithm requires the distances between the vertices and the server has to ask the proxies to take them
6. The *Session Server* makes the proxies to dynamically load the *Measurement* module
7. The *Session Server* asks the *proxies* to measure their distance to the other *proxies*. In this example, the only distance needed is $FP_1 \rightarrow FP_2$

8. The *proxies* measure their distance to the given ones
9. Each proxy sends the taken measures back to the *Session Server*
10. The *Session Server* creates the DgB-SPT (see Figure 65b)

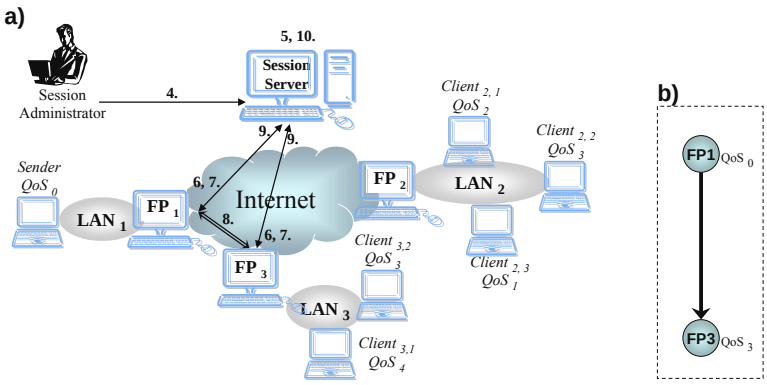

Figure 65: a) Graph creation and measurements taking; b) tree structure

11. The *Session Server* loads the *FPTP* module on the *proxies*
12. By following the DgB-SPT structure, the *Session Server* makes the *proxies* to interconnect
13. The *proxies* establish an FPTP connection
14. The *Session Server* notifies the *clients* about the session start
15. The *Session Server* notifies the *sender* about the session start. Let us remark that it is necessary to start the receivers prior to the sender.

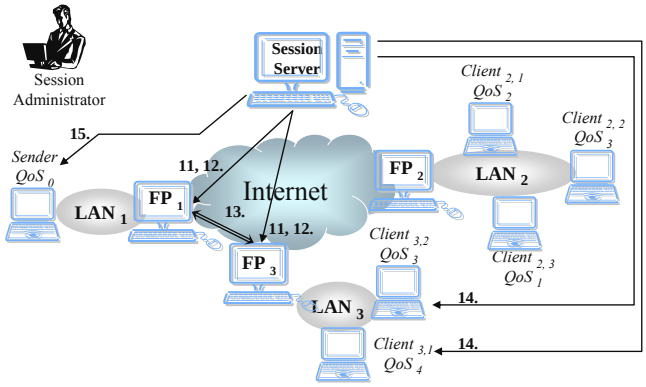

Figure 66: Proxies interconnection and clients starting

4.1.2.4. *Dynamic clients adding*

In the process of dynamic clients adding, three possible scenarios exist.

a) The new client is served by an FPTP Proxy not existing in the graph

b) The new client is served by an FPTP Proxy already existing in the session and the new client requires a lower QoS than the one received by the FP

c) The new client is served by an FPTP Proxy already existing in the session and the new client requires a higher QoS than the one received by the FP

In the first case, the new FP has to be added to the HG, and then it will be necessary to take some new distance measurements in order to complete the graph. After that, the possible tree modifications have to be transposed on the network.

In the second case, no changes are necessary, so the new client is accepted directly.

In the third case, the corresponding FP has to change its QoS and the possible tree changes have to be transposed on the network.

4.1.2.5. *First case*

The operations described in this section are illustrated in Figure 67, Figure 68 and Figure 69.

16. *Client* $_{2,1}$ sends a login message to the *Session Server*. This new client is served by an FPT not already existing in the tree
17. The *Session Server* adds the new *proxy* to the graph
18. The *Session Server* loads the *Measurement* and the *FPTP* modules on the new *proxy*
19. By following the HG structure, the *Session Server* makes the *proxy* to measure its distance to other proxies
20. The *proxy* measures its distance to the given *proxies*
21. The *proxy* sends the taken measurements back to *Session Server*
22. The *Session Server* adds the new *proxy* to the HG (see Figure 67b)

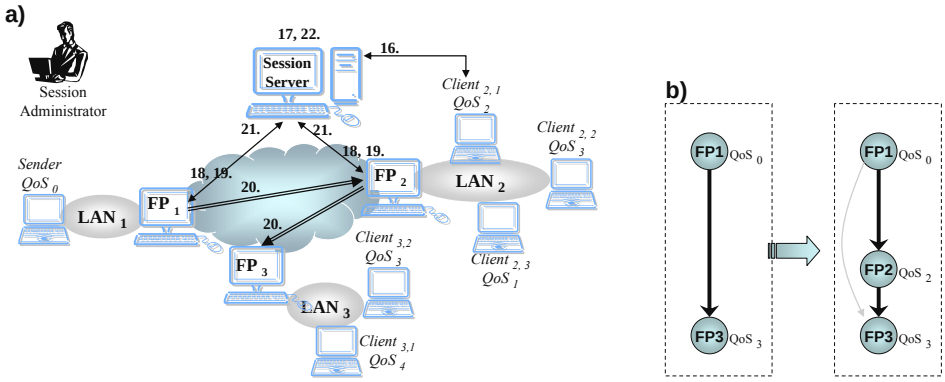

Figure 67: a) New client login and measurements taking; b) New structure for the HG

23. The S*ession Server* makes the new proxy (FP_2) to connect to its parent (see Figure 68a)
24. FP_2 establishes a new FPTP connection to its parent

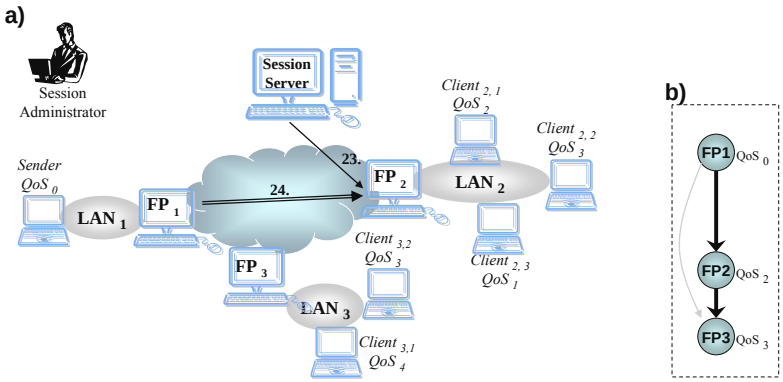

Figure 68: a) Connecting new proxy; b) new structure for the DgB-SPT

25. The *Session Server* makes FP_3 to disconnect from FP_1 (see Figure 69)
26. FP_3 disconnects from FP_1
27. The *Session Server*, by following the DgB-SPT Structure, makes FP_3 to connect to its new parent
28. FP_3 establishes an FPTP connection to its new parent
29. The *Session Server* starts the session on the new client.

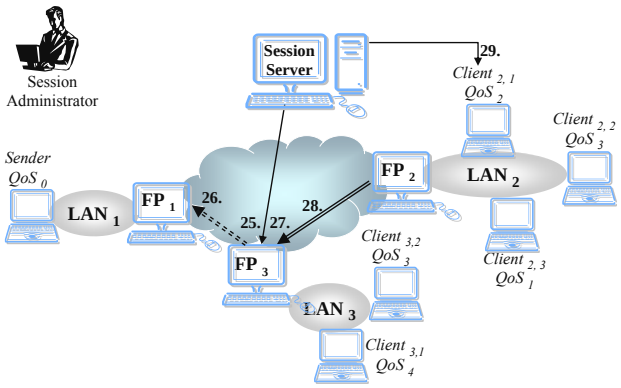

Figure 69: Proxies reconnecting

4.1.2.6. *Second case*

The operations described in this section are illustrated in Figure 70.

30. *Client* $_{2,2}$ sends a login message to the *Session Server*
31. The *Session Server* verifies the HG and the DgB-SPT. As the serving proxy already exists in the HG and the new client requires a lower QoS than the QoS received by its proxy, then the new client is accepted directly
32. The *Session Server* sends a *start session* message to the new client.

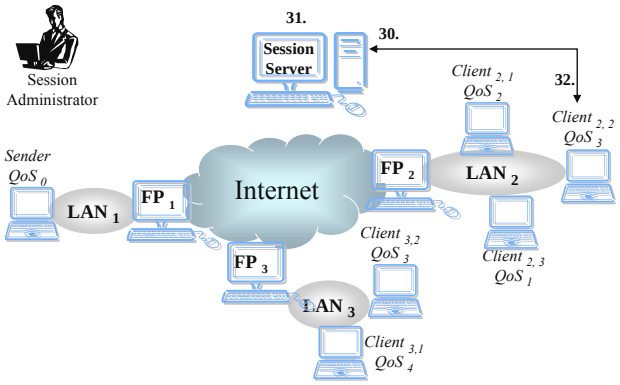

Figure 70: New client login, second case

4.1.2.7. *Third case*

The operations described in this section are illustrated in Figure 71.

33. *Client* $_{2,3}$ sends a login message to the *Session Server*
34. The *Session Server* verifies the HG and the DgB-SPT. As the serving proxy already exists in the HG and the new client requires a higher QoS than the QoS received by its proxy, then the server has to change the proxy's QoS on the HG and reconstruct the tree (see Figure 71b). All possible changes on the tree have to be transposed to the network. In this example, FP_2 changes its QoS but its parent is the same, so it is not necessary to disconnect the proxy, but only reconfigure its FPTP connection
35. The *Session Server* makes FP_2 to change its QoS parameters
36. FP_2 changes its QoS parameters
37. The *Session Server* starts the session on the new client

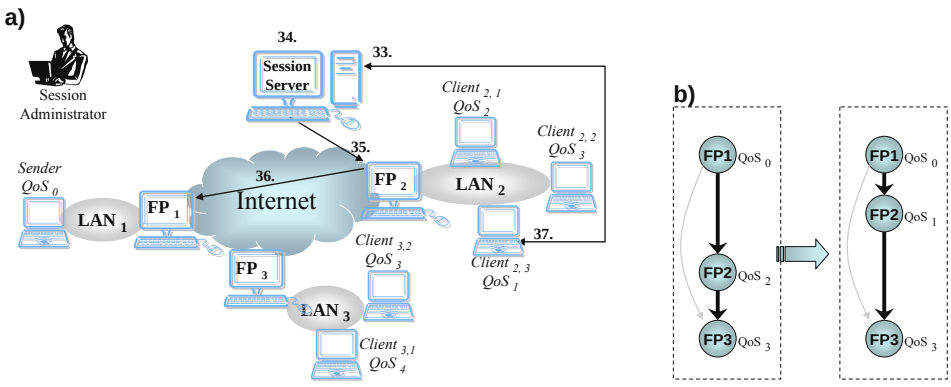

Figure 71: a) New client login and proxy's QoS changing; b) new graph structure

4.2. Global protocol architecture

Figure 72 shows the integrated protocol architecture. Figure 72a shows the *M-FPTP Proxy*. This proxy is supposed to contain a Programmable Networks Platform which will be in charge of dynamically loading the *M-FPTP* and the *Measurement* Modules. Since a programmable networks platform is out of the context of this thesis and as M-FPTP has been defined in a previous chapter, the only remaining element to be defined in our protocol is the measurement module. This module will receive from the session server a proxy address in order to measure its distance to the given proxy and will send the measure back to the server.

Figure 72b shows the *Session Server* architecture. It contains a module *SSP-QoM* which implements the entire server behavior. This module will be explained in next sections. It also contains an instance of the DgB-SPT algorithm explained in precedent chapter.

Finally, Figure 72c shows the *MM Client* architecture. In this client, the *SSP-QoM* module will be in charge of sending and receiving messages to/from the server and of instantiating the MM application.

All three elements, M-FPTP Proxy, Session Server and MM Client will be modeled by using UML. This model is exposed in the next section.

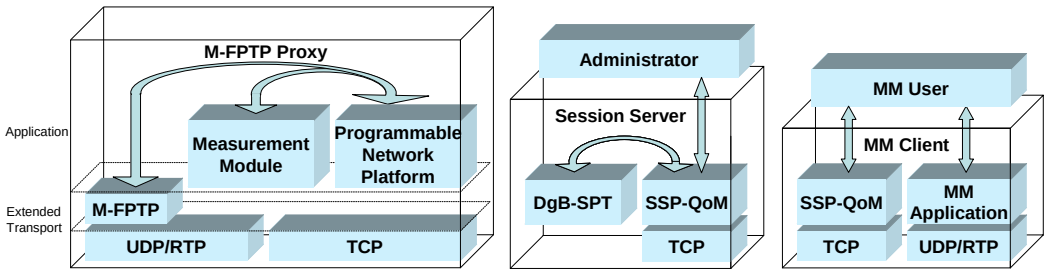

Figure 72: Protocol architecture. a) M-FPTP Proxy; b) Session Server; c) MM Client

4.3. UML Model

As said before, the proposed protocol is modeled by using UML 2.0 and SDL. This section exposes the UML model done for the requirements described in the last section. It will first describe the interfaces from the system to the user, and then it will explain the class diagram.

After that, a brief explanation of the system behavior will be given, and finally it will be shown the simulations done in order to validate the model and to verify that the proposed mode actually fulfills the requirements given in the last section.

The complete UML diagrams and their explanation can be found at author's web site [GAR-W].

4.3.1. User interface

A *usecase* UML diagram represents the interactions between the modeled system and the external elements called “actors”, i.e. the system users.

In SSP-QoM, two kinds of actors are identified: the session administrator and the multimedia users (sender and receivers). An administrator can define a session, start the defined session, stop the session, continue the session and destroy the session. The MM users can enter a session, quit the session, send and receive data.

The required parameters for all these operations are hidden in this phase of modeling. Figure 73 represents these interactions.

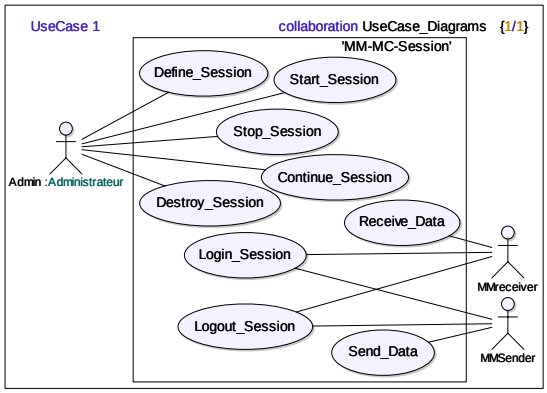

Figure 73: UseCase diagram for SSP-QoM

In order to access the defined actions, a set of messages exchanged between the system and the actors has been defined (see Figure 74).

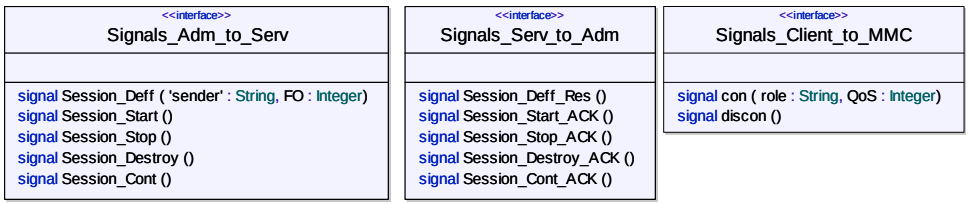

Figure 74: Messages list between the administrator, the users and the system

4.3.2. Class diagram

By following the protocol architecture shown in Figure 72, it has been created a class for representing the session: *MM_MC_Session*, and a class for each network element: *Session Server*, *FPTP Proxy* and *MM Client*. It has also been defined a set of accessory classes: *Client ID*, *ProxyID* and *MMod*. The relationships between classes are exposed in the class diagram shown in Figure 75.

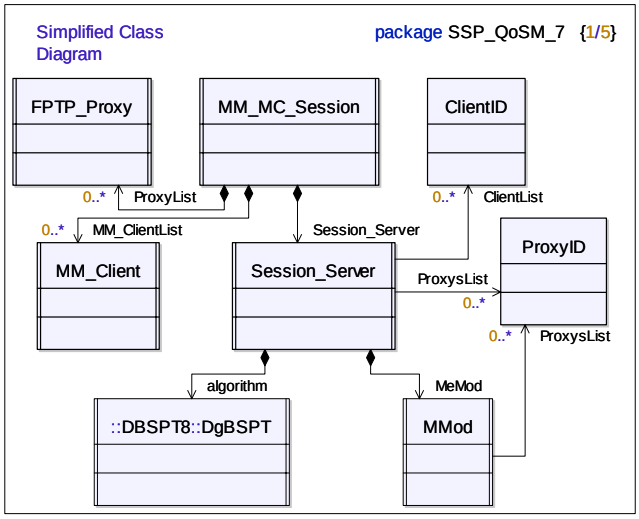

Figure 75: simplified SSP-QoM class diagram

As said before, some accessory classes have been added. These classes are shown in Figure 76.

- Class *ClientID* identifies a *multimedia client* by its ID, its corresponding server, required QoS and its role within the session
- Class *ProxyID* identifies a given proxy by its ID, its current QoS, its name and its parent's name and ID
- Class *MMod* sends the *Measurement* message to proxies and receives the distances back.

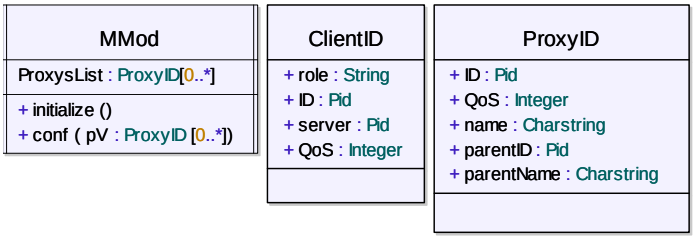

Figure 76: accessory classes in SSP-QoM UML model

4.3.2.1. *MM_MC_Session, MM_Client* and *FPTP_Proxy* classes

Class *MM_MC_Session* is a "container" which will instantiate the active classes (Server, Proxies and Clients). As shown in Figure 77, this class is composed by three attributes: a *session server*, a *proxy* list and an *MM_Client* list.

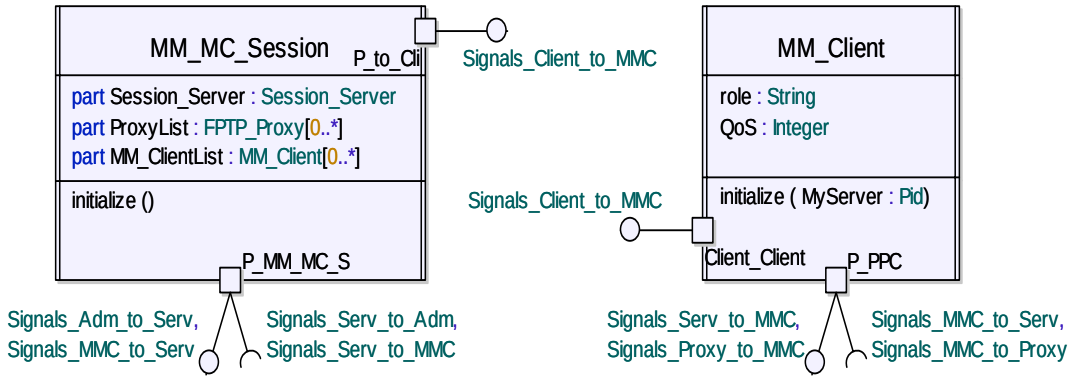

Figure 77: MM_MC_Session & MM_Client class diagram

The *MM_Client* class, its attributes and its methods are shown in Figure 77. *MM_Client* is an intermediary between the *MM Users* and the *Session Server*. It behaves as follows: it waits for a *con* message from the multimedia user and sends a *Login_Sess* request message to the S*ession Server*. After that, it can either receive a *Quit* message (and then it goes to its initial state), or a *Wait* message, or a *Start_Sess* message (and then it acknowledges the message and starts the session at the client side). The *client* can leave the session when receiving a *Quit* message from the *Session Server* or when receiving a *discon* message from the *MM User*.

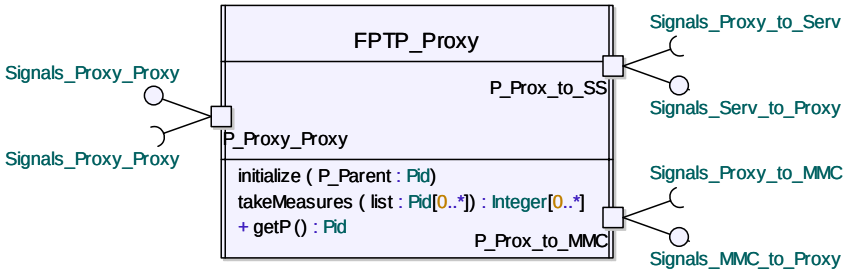

Figure 78: FPP_Proxy class diagram

The *FPTP_Proxy* class (Figure 78) contains two main procedures: *initialize* and *takeMeasures*. *Initialize* waits for *Measure* message from the *Session Server*, and then it acknowledges the message and calls the *takeMeasures* method. Finally it sends the measures taken back to the *session server* and goes to state *Ini*. Method *takeMeasures* sends a *ping* message to each given proxy, waits for the answer and finally returns the measurement taken.

4.3.2.2. *Session_Server class*

Session_Server Class (Figure 79) is the most complex class in the model. It contains a *ClientID* and a *ProxyID* lists and it instantiates the DgB-SPT algorithm and the measurement module.

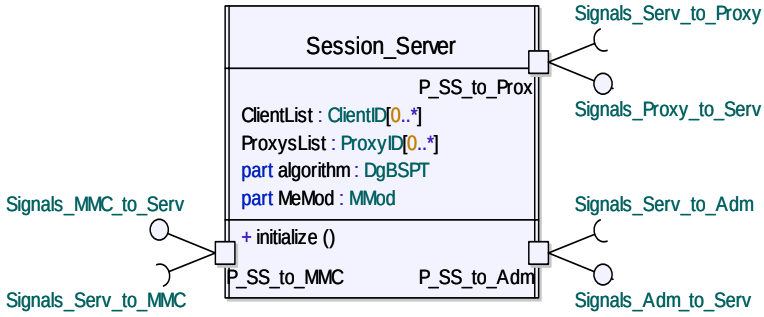

Figure 79: Session_Server class diagram

This section explains the global behavior of this class and shows, as an example of SDL capabilities, some *statechart* diagrams. The complete UML models and descriptions are accessible at authors' web site [GAR-W].

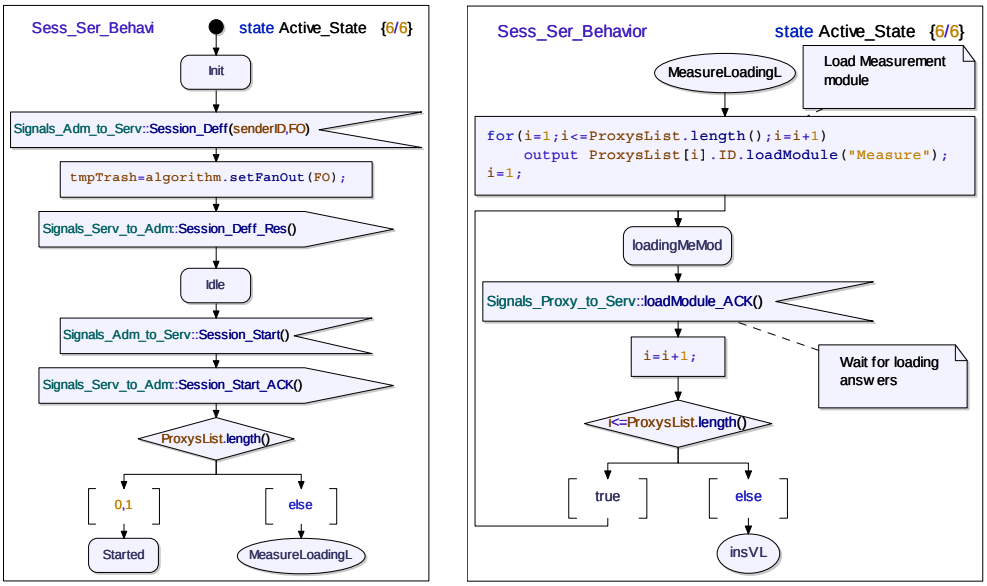

Figure 80: a) Session initialization and Session starting; b) Remote module loading

This class has three main states: *Init*, *Idle* and *started*. In the state *Init* the class is waiting for the *session definition* message from *session administrator* (Figure 80a). After having received this message, the *Session Server* sets the HG and the DgB-SPT up, acknowledges the message and goes to state *Idle*. In the state *Idle*, the *server* waits for the *login* requests from users, it stores their data and waits for *session start* message from *administrator*. Let us remark that the HG and the DgB-SPT are not created until the reception of the *session* start message from the *administrator*. When receiving the *session start* message, the *Session Server* asks the proxies to load the measurement and the FPTP modules (Figure 80b).

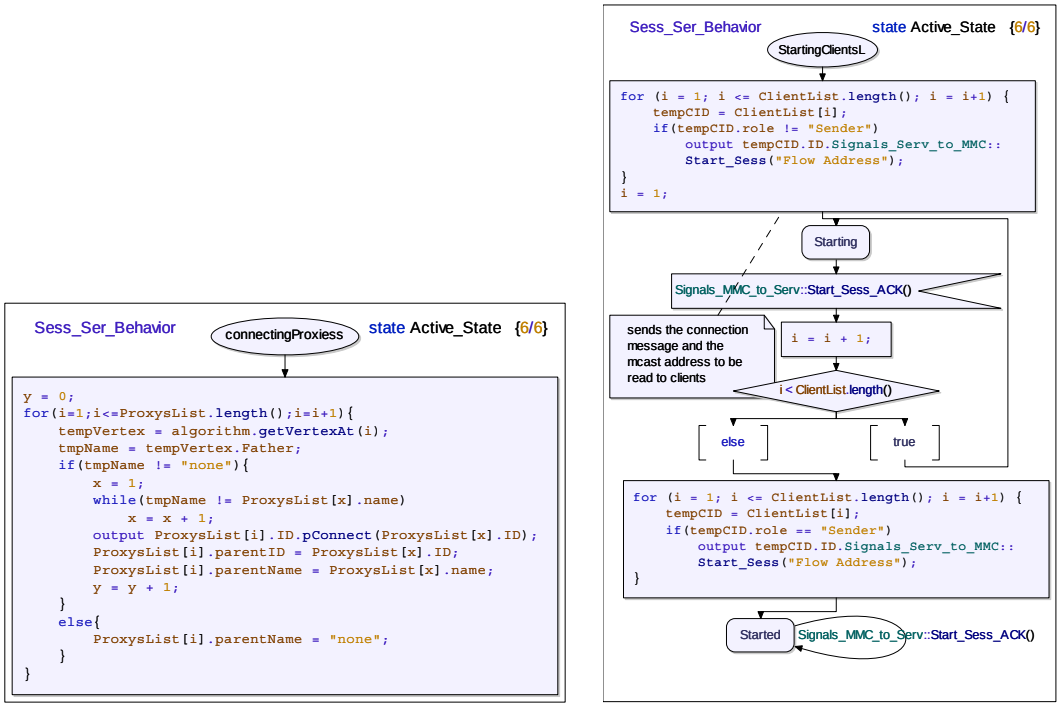

Figure 81: a) Interconnecting the proxies; b) Session starting on the clients

After remote modules loading on the proxies, the *Session Server* inserts the vertices to the HG and makes the proxies to measure their distance to other proxies, and then it creates the DgB-SPT. After, it sends the connection message to all proxies by following the tree structure (Figure 81a) and finally it sends a *connect* message containing the multicast address to be used for receiving data to all *clients* (Figure 81b).

4.3.3. System architecture

An *architecture diagram* represents the existing interfaces between the actors and the system and the organization and connections between inner elements.

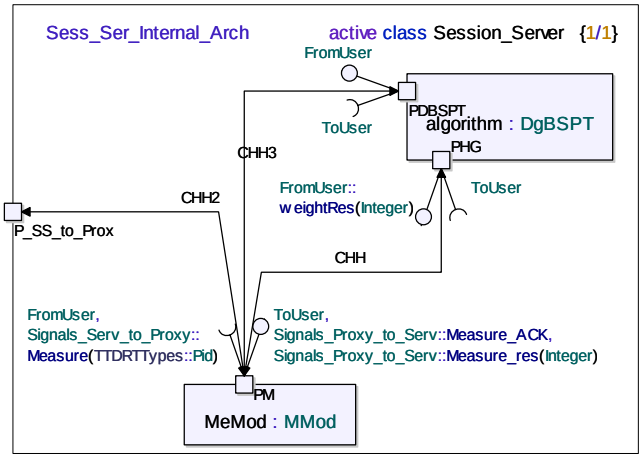

Figure 82: Session Server internal architecture

Figure 82 shows the internal structure of *Session Server*. This figure shows that the server instantiates *DgB-SPT* and *MModule*. The first module is the implementation of the algorithms shown in chapter 2 and 3 while the second module is in charge of sending the *measurement* message to the proxies.

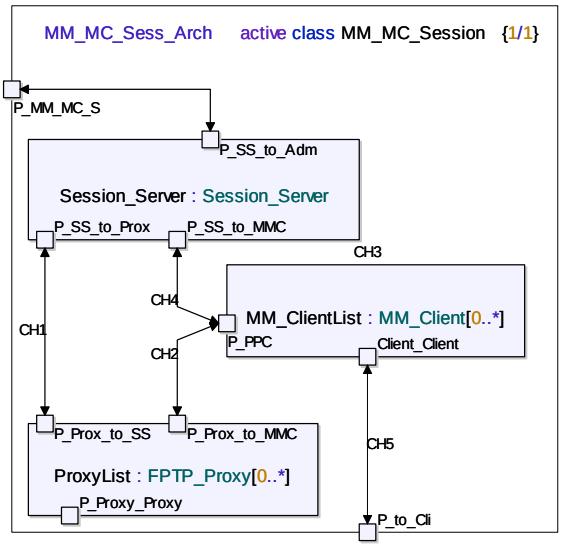

Figure 83: MM_MC_Session internal architecture

Figure 83 shows the internal structure of the *MM_MC_Session* class. This class instantiates the *Session Server* class such as a list of *MM_Clients* and a list of *FPTP Proxies*. The *Session Server* is connected to the two lists and the list of clients is dynamically connected to the list of proxies.

Figure 84 first shows the list of messages transmitted from *multimedia client* to *session server*. Apart from acknowledgement messages and logout messages, a *multimedia client* can send a *login_Sess* message which contains the ID of its corresponding proxy, its role within the session (for example *sender* or *receiver*), and the desired QoS.

The same figure gives the messages going from *session server* to *multimedia clients*. These messages are: *logout_Session_ACK* used to confirm a logout request, *wait* used to inform the client that either the session is not yet started or the session is paused, *Start_Sess* used to inform the client about the local multicast address used for data sending/receiving, *Session_Cont* used to notify that session is not more paused, and *quit*.

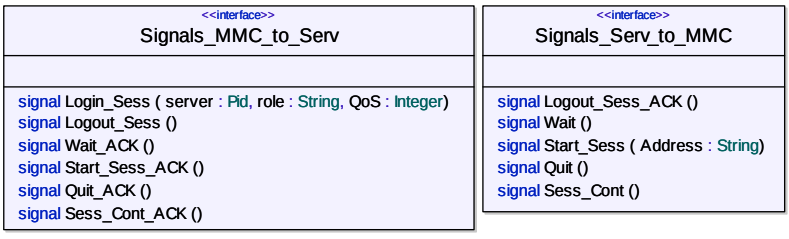

Figure 84: list of messages between MMC and Session Server

Figure 85 shows three messages list. The first one contains the messages going from the *session server* to the *proxies*. These messages are: *measure* containing a list of proxies'

addresses to which measure distance, *pConnect* containing a proxy address to be used as "parent" within the multicast tree, and *loadModule* used to dynamically load either *FPTP* or *measurement* module.

The second list contains the messages going from *FPTP_proxy* to *session server*. All messages, but one, are used to acknowledge the received messages. In the case of *Measure_res*, this message contains a list of distance to other proxies.

The third list shows the messages interchanged among proxies. The *ping* message measures the distance between two proxies and it is answered with *ping_res* message. Then *FPTP_Connect* message is used to connect two proxies; this message is answered by the *FPTP_Connect_ACK* message and so on.

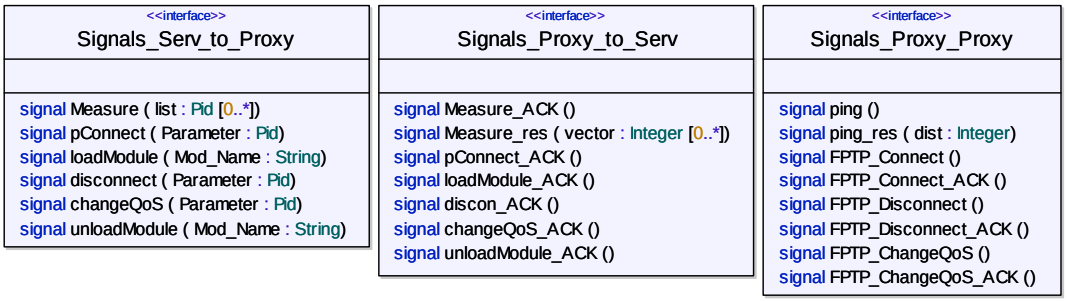

Figure 85: list of messages between Session Server and Proxies

Finally, Figure 86 shows the messages interchanged between *session administrator* and *session server*. All messages sent by *session administrator* are answered by *session server* with an *ACK* message.

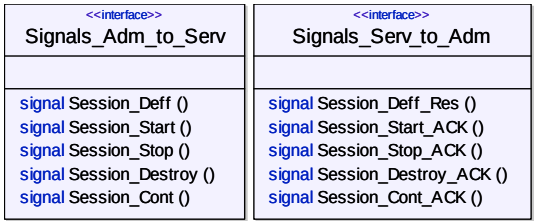

Figure 86: list of messages between Session Administrator and Session Server

4.4. Step by step model validation

The used CASE tool, TAU G2, permits to define the desired behavior as a set of *finite state machines* (SDL *statecharts*). Then, this behavior can be simulated and the results are shown in the form of *scenarios*, also called *sequence diagrams* in UML 2. This simulation allows the validation of the UML model definition, its behavior and its interaction; it permits also to verify if the model corresponds to the specified requirements.

This section exposes a simulation done in order to validate the correct functioning of the protocol model exposed previously. The simulation targets to obtain the behavior defined in section 4.1.2. So, the network is configured as follows:

- A *Session_Server*
- Three proxies: *ProxySend*, *Proxy1* and *Proxy2*
- Four clients:
    - *MMSender*, connected to *ProxySend*
    - *MMClient1* and *MMClient2* connected to *Proxy1*
    - *Client3* connected to *Proxy2*

The required QoS for each client are defined as follows:

- *MMClient1* has QoS 1
- *MMClient2* has QoS 2
- *MMClient3* has QoS 3

In order to verify the DgB-SPT properties in a small example and with a little number of proxies, the maximal fan-out limit is set to 1.

Diagram in Figure 87 shows the different steps followed in this simulation. In the first step, *MM_MC_Session* class instantiates the server, the proxies and the clients. At this moment, every client knows its corresponding proxy.

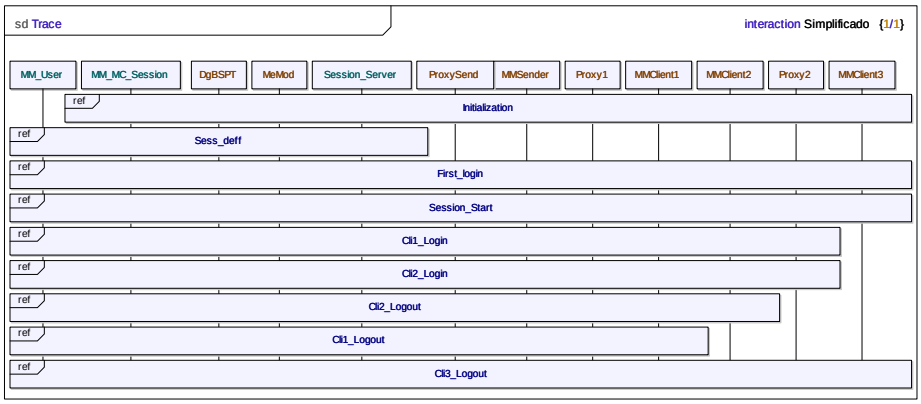

Figure 87: Simplified sequence diagram

After system initialization, the only possible action is the *session definition* from the *administrator*. This message states the sender and the fan-out limit.

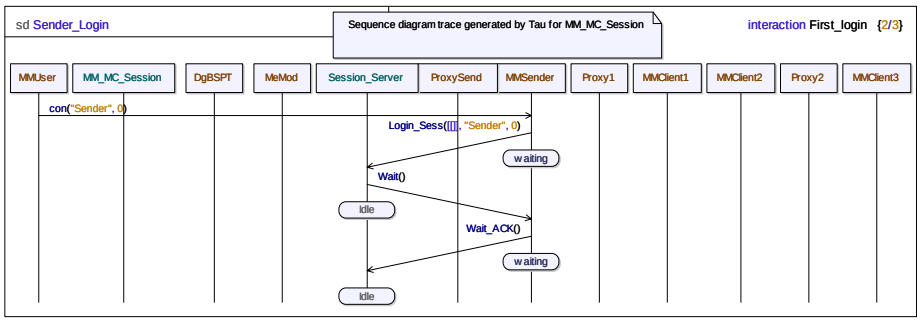

Figure 88: Sender login

Prior to session starting, two clients log into the session: the *sender* and *MMClient3*. Figure 88 shows the interchanged messages logging the *sender into the session*. In this simulation, the messages coming from the *administrator* and from *MM user* are sent manually, all the others are sent automatically by following the behavior defined in SDL. When receiving the *Login_session* message from *MMSender*, the *Session Server* sends a *Wait* message back to the sender and continues waiting for other clients to login. The *MMSender* acknowledges the message and goes to state *Waiting*. The login process of *MMClient3* follows the schema given Figure 88.

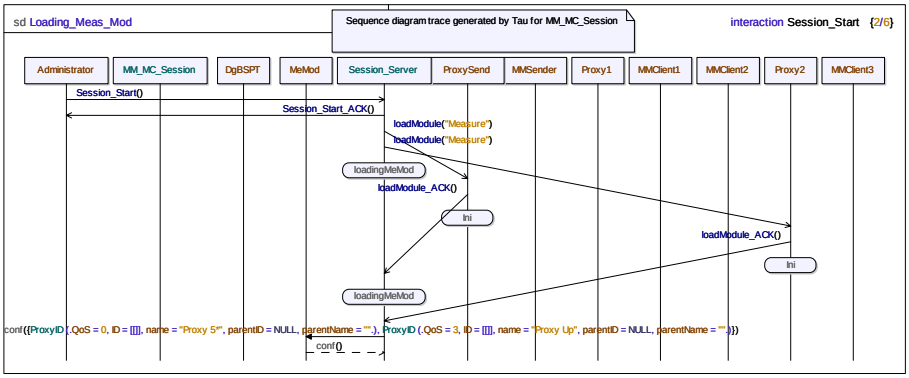

Figure 89: Measurement module loading

After *MMSender* and *MMClient3* login, the *Administrator* starts the session. After receiving the *Start_Session* message, the server makes *ProxySend* and *Proxy1* to load the *Measurement* module and starts creating the HG and the DgB-SPT (Figure 89).

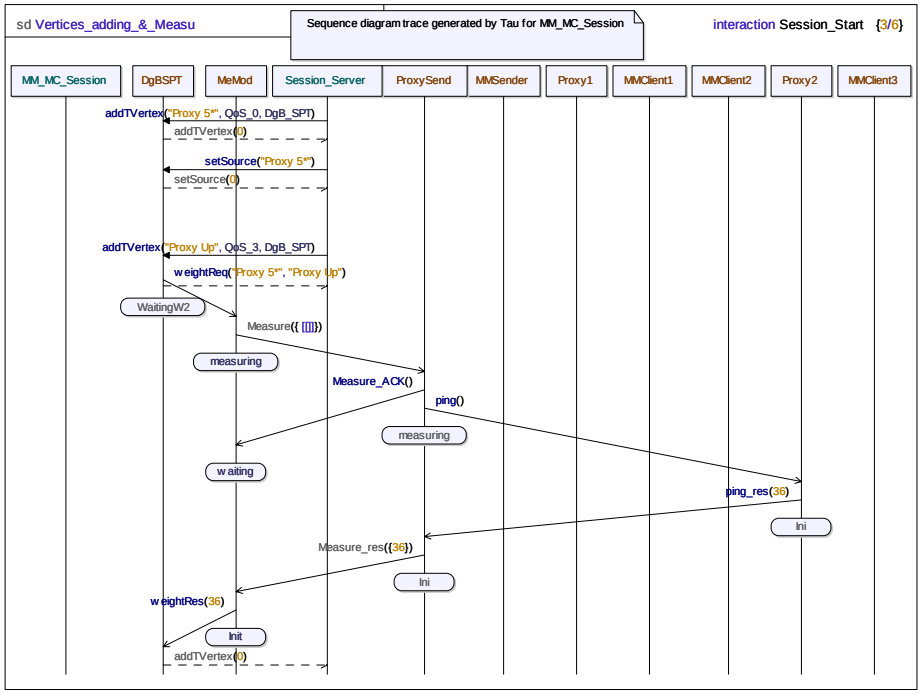

Figure 90: Vertex adding and distance measurement

For each edge added to the HG, the *Measurement Module* in the server asks the distance to the corresponding proxy. This proxy measures its distance to the given proxy and sends the measurement back (Figure 90). After collecting all necessary distances, the server makes the participating proxies to load the *FPTP module* (this procedure is similar to Figure 89)

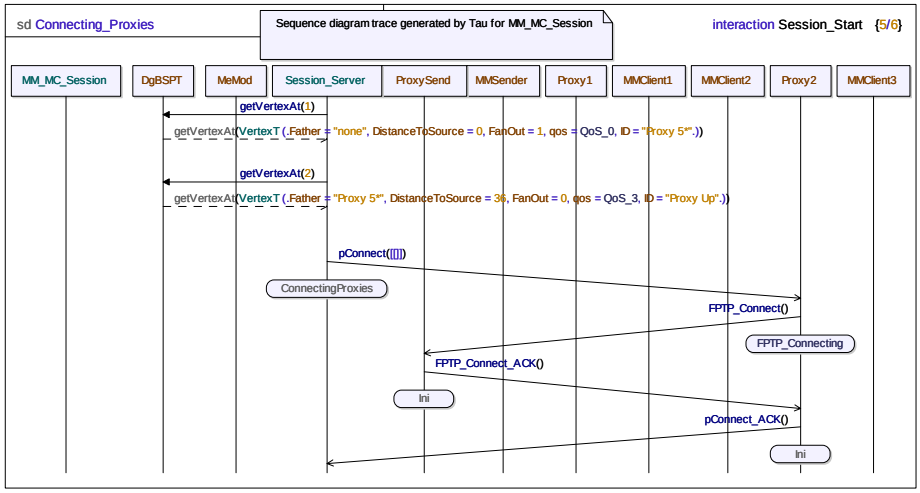

Figure 91: Proxies connection

Once loaded the *FPTP module* on the participating proxies, the server sends a *proxyConnect* message to proxies by following the tree structure. When receiving this message, a proxy establishes an FPTP connection to its corresponding parent and sends a *proxyConnect acknowledge* message to the sender (Figure 91).

After connecting all the proxies to their corresponding parent, the server notifies the clients that session has been started, and finally it notifies the *MMSender* to start sending data to the session.

As explained in section 4.1.2.4, there are three possible scenarios when adding a new client to a session. In the next step of this simulation a new client logs into the session and its corresponding proxy is not present in the HG.

MMClient3 sends a *Login_Session* message to the server and it adds the new proxy to the HG, collects the required distances and modifies the tree. In this new tree structure, *Proxy1* has *ProxySend* as parent and *Proxy2* has to change its parent from *ProxySend* to *Proxy1*. The server has to reflect these changes into the network, and then it makes *Proxy1* to establish an FPTP connection to *ProxySend*. After that the server it has to make Proxy2 to disconnect from *ProxySend* and to reconnect to *Proxy1* (Figure 92). Let us remark that the process of reconnecting the already connected proxies is done at the very end in order to interrupt as few as possible the data flow for the previous connected clients. Finally, the server informs the new client about the correct login and the local multicast address to be read.

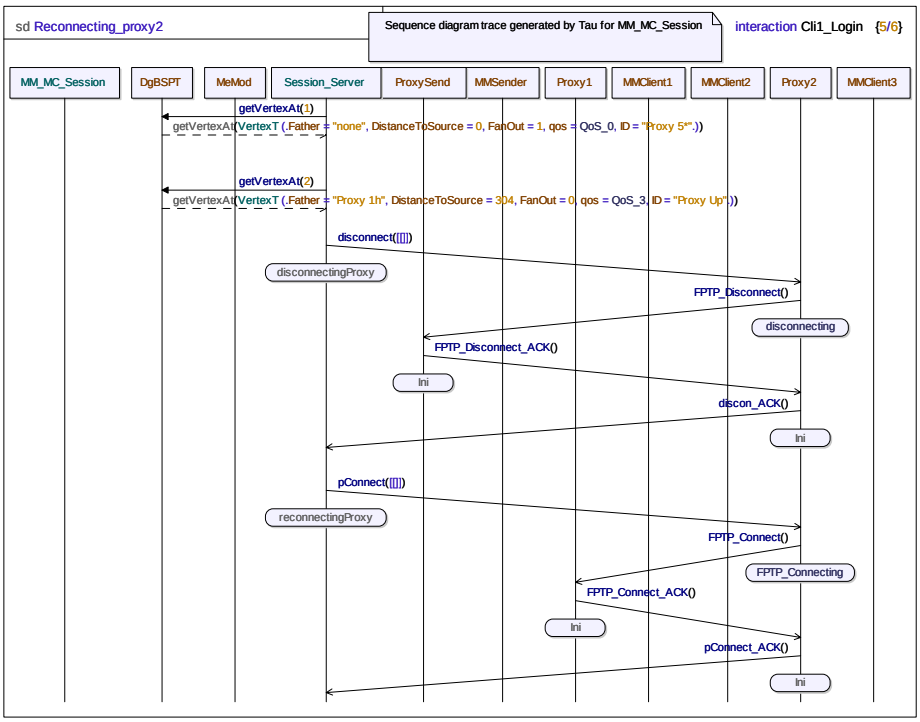

Figure 92: Reconnecting Proxy 2

In the next step of this simulation, a new client (*MMClient2*) logs into the session and its corresponding proxy (*Proxy1*) already exists in the tree, but the client requires a higher QoS than the one received by its proxy; so the server has to increment the QoS received by the proxy and maybe reconfigure the tree structure.

The QoS upgrading is seen by the server as deleting the vertex from the HG and the tree and re-inserting it with a new QoS. Let us remark that none of these changes is reflected to the network but at the end of the operation in order to interrupt the connected clients as few as possible.

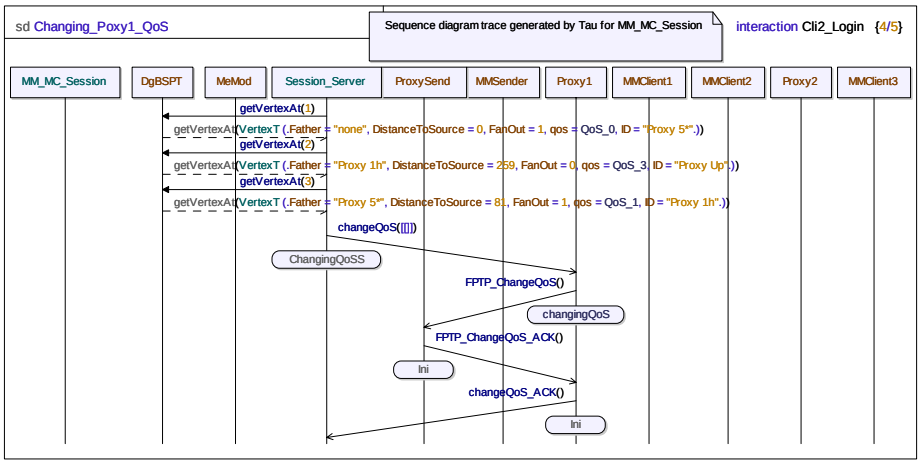

Figure 93: Changing proxy's QoS

In this simulation, at the end of the re-insertion process, *Proxy1* has changed its QoS but its parent has not changed, so, it only changes its QoS with its current parent (Figure 93).

At this moment, all the defined clients have logged into the session. Now, the simulation continues by logging them out from the session.

The first client to logout is *MMClient2*. As its proxy still has some clients in its served LAN, no changes are done to the HG and the tree and the client is simply deleted from the client list.

Then, *MMClient1* logs out from the session. In this case, *Proxy1* has no more clients in its served LAN, but it still is the parent of *Proxy2*. In order to interrupt the data flow as few as possible, it has been decided that a proxy stays in the session (and so in the HG and in the tree) as long as it has a client in its LAN or it is parent of another proxy. So, no changes are necessary in the HG and the tree and the client is simply deleted from the client list.

In the last step of this simulation, MM*Client3* logs out from the session. In this case, the server realizes that *Proxy2* has no more clients in its LAN and has no children in the tree, so the server disconnects it from the session and makes it to unload the unused modules (*Measurement* and *FPTP* modules). By doing this, *Proxy1* keeps with no children and no clients, so the server disconnects it from the session and makes it to unload the unused modules. The disconnection of *Proxy1* is shown in Figure 94.

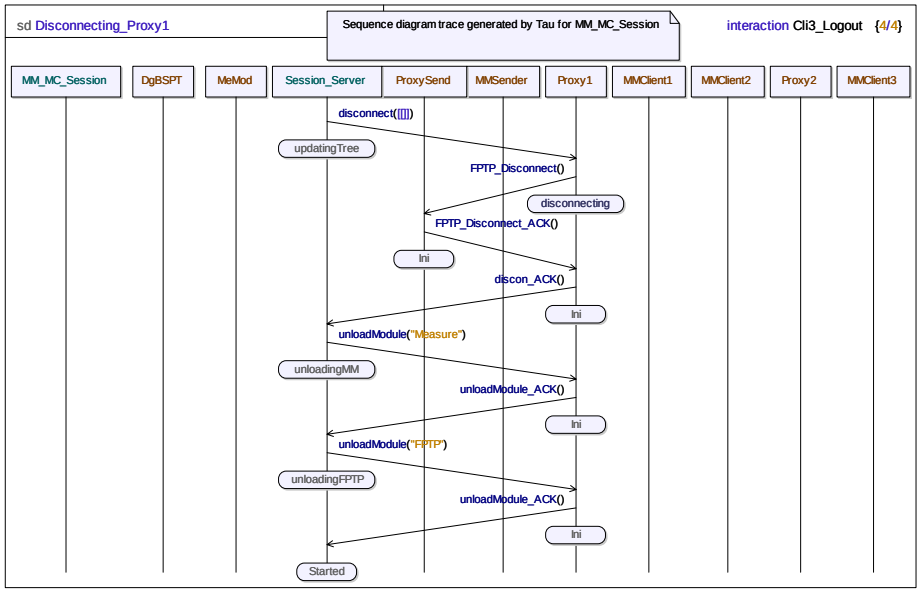

Figure 94: Proxy disconnection process

Let us remark that, in the case of *Sender* logout, the entire session is finished.

The simulation performed shows that the defined SSP-QoM UML model behaves just as expected.

4.5. Partial conclusions

In the debugging tests shown in preceding section, it was the programmer who decided when a given client had to login or out. This kind of simple simulation has permitted to test some particular aspects of the protocol behavior on small simulations. The tested behavior was:

- Proxy adaptation to clients' QoS. As said before, in SSP-QoM, the QoS assigned to a given proxy is higher than or equal to the maximal QoS of any of its served clients. If a new client asks for a QoS higher than the one received for its serving proxy, the proxy has to increment its QoS. This change can imply some modifications to the tree structure and these modifications have to be reflected on the network. This behavior has been tested and validated.

- Proxies insertion. When a proxy is added to the session, the tree structure can change and some already connected proxies can change of parents. These modifications have to be reflected on the network by disturbing the clients as few as possible. For doing so, the network modifications are done at the end of the logical tree reconfiguration. This behavior has been tested and validated.

- Proxies deletion. When a proxy is deleted from the tree, a still connected proxy can be "approached" to the source. Another possible consequence of proxy deletion process is that, as explained before, a set of proxies have to be also deleted from the tree. These modifications have to be reflected in the network. This behavior has been tested and validated.

4.6. Extended tests

In order to verify and validate the global protocol behavior, a set of extended tests have been defined and performed. In these tests, it has been created a new class representing the users' behavior. This class randomly logs in and out the users and ends the session. For these tests, a new network configuration has been defined. This configuration is formed by:

- 40 clients: *Sender*, $Client_1$, …, $Client_{39}$
- 10 proxies: *ProxySender*, $Proxy_1$, …, $Proxy_9$

The clients are organized as follows:

- *Sender*, $Client_1$, … and $Client_3$ are served by *ProxySender*
- $Client_4$, … and $Client_7$ are served by $Proxy_1$
- $Client_8$, … and $Client1_1$ are served by $Proxy_2$
- Etc.

The QoS for each client is randomly assigned by the *User* class. This class connects all the clients in a random order and then randomly generates login/logout on clients. Finally, the User logs all the clients out of the session.

The maximal fan-out limit has been defined to 4.

The user has been programmed to perform 200 login/logout operations.

The goals of these tests have been:

- Analyze the HG structure on a highly dynamic session.
- Verify the tree properties on a highly dynamic session.
- Show the robustness of the protocol on a highly dynamic session.

These tests have shown that our models behave as expected:

- It was verified that at the end of the tests, most of the proxies were assigned a high QoS; this can be easily explained because, by selecting the higher QoS, the only scenario where it decrements is when deleting the proxy.

- The output-degree constraints in the tree were kept on all the performed tests, i.e. no vertex had a higher output degree than the defined one.

- It was verified that the protocol can correctly manage sessions with a big number of logins and logouts and the protocol behavior was validated.

4.7. Chapter Summary and Discussion

This chapter has shown the work done in order to integrate and validate the solutions proposed in previous chapters i.e. the M-FPTP proxies, the HG and the DgB-SPT. It has also shown the feasibility of the integration of these solutions and models and reveals the necessity of session protocols and mechanisms adapted to user QoS. For doing so, this chapter has proposed a new session protocol called Simple Session Protocol for QoS Multicast (SSP-QoM). It is composed by three elements: a session server, a set of FPTP proxies and a set of multimedia clients. The Session server is the key element in the protocol: it receives the login/out requests from clients, their QoS requirement and their proxy ID; it creates the HG and the DgB-SPT. This server is also charged of making the proxies to dynamically load a distance measurement module and the M-FPTP module.

The protocol requirements and its expected behavior have been exposed in section 4.1 while section 4.2 has exposed the protocol architecture. Then, section 4.3 has shown the UML protocol model, its interfaces, the interchanged messages with the user, its class diagram and finally its architecture diagram. Then, the protocol model validation has been done by simulating the UML module.

The proposed protocol can use a programmable networks platform on the FPTP Proxies in order to dynamically load the *measurement* and the *FPTP* modules. In the case where no programmable networks platform is present on the proxies, without changing the protocol behavior, the proxies can permanently have these modules. In this case, the corresponding *loadModule* and *unloadModle* messages and their corresponding acknowledgements messages have simply to be eliminated from the UML model.

Conclusions and Further Work

It has been shown in a first chapter that many lacks still exist in IP multicast which hinders its development, its deployment and its ubiquity. It has also been explained the necessity of new multicast mechanisms and algorithms oriented to QoS. Then, it has been explained the needs of systems modeling, validation and verification and it has been given an introduction to the methodology and language used in this dissertation, MDA and UML 2.0.

It has next been proposed some modifications to the FPTP proxy architecture in order to extend its capacities to multicast communications. This new proxy architecture, called M-FPTP, permits to relay many local multicast networks through single FPTP connections and to define a different QoS for each one. This new configuration is based on the ALM technology. In order to solve the problems originated by the all-to-all policy of ALM, it has then proposed a new network topology model which takes into account at the same time the network performances and the different users QoS requirements. This new configuration, called Hierarchized Graph due to its organization in QoS hierarchies, permits to optimize the network resources. The HG provides a model well adapted to a differentiated QoS oriented multicast.

Next, it has been shown that the application of standard SPT algorithms on a graph such as an HG can lead to overloading problems on the source. It has then been proposed an algorithm called Degree Bounded Shortest Path Tree which finds a spanning tree where the maximal output degree on each vertex is bounded and where the distance-to-the-source for all the vertices is minimized. This algorithm has been modeled by using UML 2.0. The obtained model has been simulated and the results have been compared with Dijkstra's algorithm. These tests have shown that it is possible to limit the maximal output degree on the vertices without changing considerably the distance-to-the-source minimization performance of a standard SPT.

Finally, the last chapter of this dissertation has proposed a session protocol called Single Session Protocol for QoS Multicast (SSP-QoM) which integrates the M-FPTP, the HG and the DgB-SPT models. SSP-QoM collects the user's login and logout requests and their QoS requirements; it also collects the proxies IDs and stores their corresponding QoSs and clients list. This protocol measures the distance between proxies and then creates an HG and a DgB-SPT. Next, SSP-QoM dynamically deploys the M-FPTP sender and receiver proxies by following the tree structure. The proposed protocol has been modeled by using UML 2.0 and then the model has been simulated in order to verify its compliance with the protocol specification. These models and tests have demonstrated the feasibility of a service providing a differentiated-QoS-oriented multimedia multicast service.

Without a doubt, one of the first perspectives emanating of this dissertation is the implementation and the deployment of all the elements in order to test the solution into a real network. At this time, a Java API of the HG and the DgB-SPT is available; this API has been used in order to test the tree's performances. Nevertheless, the SSP-QoM protocol has not been implemented. However, based on the UML model, this implementation should be done easily.

In this work, the distance measures taken into account were the RTT because of its facility to be obtained. However, as said before, the distance between nodes can be extended to take into account the bandwidth, the delay, a given priority, etc. It will be an interesting add-in to this work to define a more complete definition of nodes distance. This definition could be configurable from the session definition in order, for example, to give more priority to one attribute than to another.

The current version of SSP-QoM has defined that the QoS assigned to a given proxy is higher than or equal to the maximal QoS of any of its served clients. This policy permits to satisfy the most demanding clients but can penalize those clients with small performances. Another weakness of this policy is that, in a paid service, some clients will be receiving a better QoS than the one they have paid for. Another possible policy is to define that the QoS assigned to a given proxy is equal to the minimal QoS of any of its served clients. This new policy avoids the overloading problems on those clients with low performance. However, even if this policy allows all clients to receive the multimedia flow, it limits the high performing client QoS in order to favor the weakest ones. An interesting extension to our work is to add the possibility to choose different desired policies. This parameterization can be done at session definition.

As a direct extension of this work, an interesting perspective is to create an API to be used by a specialized session protocol.

Finally, the present version of SSP-QoM has been defined to be simple, so other capabilities could be added. For instance, security and reliability have to be added in terms of new algorithms and in terms of session attributes, including authentication.

Bibliography

[ADA97] ADAM C.M., LAZAR A.A., LIM K.-S., and MARCONCINI F., "The Binding Interface Base Specification Revision 2.0", *OPENSIG Workshop on Open Signalling for ATM, Internet and Mobile Networks,* Cambridge, UK, April 1997.

[ALE98] ALEXANDER D.S., ARBAUGH W.A., HICKS M.A., KAKKAR P., KEROMYTIS A., MOORE J.T., NETTLES S.M., and SMITH J.M., "The SwitchWare Active Network Architecture", *IEEE Network Special Issue on Active and Controllable Networks, vol. 12 no. 3*, 1998.

[ALM00] ALMEROTH K., "The Evolution of Multicast: From the MBone to Inter-Domain Multicast to Internet2 Deployment", *IEEE Network Special Issue on Multicasting*, January/February 2000

[AME94] AMER P. D., CHASSOT C., CONNOLLY T. J., DIAZ M., and CONRAD P., "Partial-order transport service for multimedia and other applications", *IEEE/ ACM Transactions on Networking*, 2(5):440--456, 1994

[ANG98] ANGIN O., CAMPBELL A.T., KOUNAVIS M.E., and LIAO R.R.-F., "The Mobiware Toolkit: Programmable Support for Adaptive Mobile Networking", IEEE Personal Communications Magazine, Special Issue on Adaptive Mobile Systems, August 1998.

[BAL95] BALLARDIE T., FRANCIS P. and CROWCROFT J., "Core based trees CBT: An architecture for scalable multicast routing", *ACM Sigcomm*, San Francisco California, USA, pp. 85-95 September 1995

[BAU95] BAUER F. and VARMA A., "Degree-Constrained Multicasting in Point-to-Point Networks", *Proceedings of IEEE INFOCOM '95*, April 1995

[BIA03] BIANCHI G., BLEFARI-MELAZZI N., BONAFEDE G. and TINTINELLI E., "QUASIMODO: Quality of service-aware multicasting over diffserv and overlay networks", *IEEE Network (Special Issue on Multicasting: An Enabling Technology*), pp. 38-45, Jan.-Feb. 2003, Vol.17 No.1.

[BJO00] BJÖRKANDER M., "Graphical Programming using UML and SDL"; TELELOGIC documentation resources, http://www.telelogic.com, december 2000

[BJO02] BJÖRKANDER M., "Model-Driven Development and UML 2.0", White Paper, TELELOGIC documentation resources, http://www.telelogic.com

[BLA98] BLAKE, S., et. al., "An Architecture for Differentiated Services", RFC 2475, December 1998.

[BOE81] BOEHM B. W., "Software Engineering Economics", *Prentice Hall,* 1981

[BOR01] BORDER J., KOJO M., GRINER J., MONTENEGRO G., SHELBY Z., "Performance Enhancing Proxys Intended to Mitigate Link-Related Degradations", RFC 3135, June 2001

[BRA94] BRADEN, B., et. al., "Integrated Services in the Internet Architecture: an Overview", RFC 1633, June 1994.

[CAL01] CALVERT K. L., CAMPBELL A. T., LAZAR A. A., WETHERALL D., YAVATKAR R., "Active and Programmable Networks", *Guest Editorial, IEEE Journal on Selected Areas in Communications, Vol 19, No. 3*, March 2001

[CAM96] CAMPBELL A., "A Quality of Service Architecture", Thesis submitted for the degree of Doctor of Philosophy, January 1996

[CHA96] CHAN M.-C., HUARD J.-F., LAZAR A.A., and LIM K.-S., "On Realizing a Broadband Kernel for Multimedia Networks", *3rd COST 237 Workshop on Multimedia Telecommunications and pplications*, Barcelona, Spain, November 25-27, 1996.

[CHA00] CHAWATHE Y., MCCANNE S. and BREWER E., "An architecture for Internet content distribution as an infrastructure service", (February 2000) http://www.cs.berkeley.edu/~yatin/papers

[CHA03] CHALMERS R. C., ALMEROTH K. C., "On the topology of multicast trees", *IEEE/ACM Transactions on Networking*, no. 1, Feb 2003 pp. 153-165

[CHU00] CHU Y., RAO S. G. and ZHANG H., "A case for end system multicast"; *Proc. ACM SIGMETRICS*, June 2000

[CLA90] CLARK D. and TENNENHOUSE D., "Architectural considerations for a new generation of protocols", *Proceedings ACM SIGCOMM*, pages 200-208, September 1990

[COR90] CORMEN T. H., "Introduction to Algorithms", *MIT Press* 1990; ISBN 02620311418

[DEE90] DEERING S. and CHERITON D. R., "Multicast routing in datagram internetworks and extended LANs", *ACM Transactions on Computer Systems*, May 1990

[DEE96] DEERING S., ESTRIN D., FARINACCI D., JACOBSON V., LIU G. and WEI L., "PIM architecture for wide area multicast routing", *IEEE/ACM Transactions on Networking*, pp. 153-162, Apr 1996

[DIA94] DIAZ M., CHASSOT C. and LOZES A., "From the Partial Order Connection Concept to Partial Order Multimédia Connections", *Proceedings of HIPPARCH Workshop*, Dec. 1994

[DIJ59] DIJKSTRA E. W., "A Note on Two Problems in Connection with Graphs", *Numer, Math.*, vol. 1, 1959, pp. 269-71

[ERI94] ERIKSSON H., "The Multicast Backbone"; *Communications of the ACM*, vol. 8, pp. 54-60, 1994

[EST98] ESTRIN D., FARINACCI D., HELMY A., THALER D., DEERING S., HANDLEY M., JACOBSON V., LIU C., SHARMA P. and WEI L., "Protocol independent multicast sparse mode PIM-SM: Protocol specification", Internet Engineering Task Force IETF, RFC 2362, June 1998

[EXP03] EXPOSITO E., "Specification and implementation of a QoS oriented transport protocol for multimédia applications", PhD dissertation, Institut National Polytechnique de Toulouse. December 2003, Toulouse, France.

[FRI00] FRIGIONI D., MARCHETTI-SPACCAMELA A. and NANNI U., "Fully dynamic algorithms for maintaining shortest paths trees", *Journal of Algorithms*, v.34 n.2, p.251-281, Feb. 2000

[GAN99] GANG F. and TAK SHING P. Y., "Efficient Multicast Routing with Delay Constraints" , *International Journal of Communication Systems*, Vol. 12, Issue.3, May/June 1999, pp.181 – 195

[GAR01-2] GARY T. et al, "A configurable and extensible transport protocol", *INFOCOM*, 2001: 319-328.

[GAR-W] Author's web site: http://www.laas.fr/~dgarduno

[GCAP] GCAP: Global Communication Architecture and Protocols, IST-1999-10 504, home site: http://www.laas.fr/GCAP/

[HUT91] HUTCHINSON N., PETERSON L., "The x-kernel: An architecture for implementing network protocols", *IEEE Transactions Software Engineering*, vol. 17, no. 1, 1991.

[IEC] International Engineering Consortium, site: http://www.iec.org

[ITO02] ITO H., NAGAMOCHI H., SUGIYAMA Y., FUJITA M., "File transfer tree problems", *Lecture Notes in Computer Science*, vol. 2518, Springer-Verlag, 2002, pp.441--452

[KNU01] KNUTSSON B., "Architectures for Application Transparent Proxys: A Study of Network Enhancing Software", *DoCS* 01/118, 119 pp. Uppsala. ISSN 0283-0574, May 2001

[KOB03] KOBRYN C, SAMUELSSON E., "Drivig Architectures with UML 2.0; The TAU G2 Approach to Model Driven Architecture", White Paper, TELELOGIC documentation resources, http://www.telelogic.com

[KOH02] KOHLER E. et al, "Datagram Congestion Control Protocol (DCCP)", Internet Draft : draft-kohler-dcp-04.txt, Octobre 2002

[KRU56] KRUSKAL J.B., "On the shortest spanning subtree of a graph and traveling salesman problem", *Proc Amer. Math. Soc*. 7 (1956) pages 48-50

[KRU96] KRUMKE S.O., NOLTEMEIER H., MARATHE M.V., RAVI S.S. and RAVI R., "Improving Steiner Trees of a Network under Multiple Constraints", Technical Report LA-UR 96-1467, Los Alamos National Laboratory, Los Alamos, NM, 1996

[KUL98] KULKARNI A.B. MINDEN G.J., HIL, R., WIJATA Y., GOPINATH A., SHETH S., WAHHAB F., PINDI H., and NAGARAJAN A., "Implementation of a Prototype Active Network", *First International Conference on Open Architectures and Network Programming (OPENARCH),* San Francisco, 1998.

[LEE01] LEE K, HA S, LI J, et al., “An application-level multicast architecture for multimedia communications”, *Proceedings of the 8th ACM International Conference on Multimedia*, Los Angeles, USA, 2001: 398-400

[LAX00] LAXMAN H. SAHASRABUDDHE, BISWANATH MUKHERJEE, "Multicast Routing Algorithms and Protocols: A Tutorial", *IEEE Network*, January/February 2000, pp.90-102

[LOR02] LORENZ D. H. and ORDA A., "Optimal partition of QoS requirements on unicast paths and multicast trees", *IEEE/ACM Transactions on Networking*, no. 1, Feb 2002 pp. 102-114

[MAT01] MATHY L., CANONICO R., and HUTCHISON D., "An Overlay Tree Building Control Protocol", *3rd Int'l. Wksp. Networked Group Commun*., London, U.K., Nov. 2001

[MDA] http://www.OMG.org/mda

[MOY94] MOY J., “Multicast extensions to OSPF”, Internet Engineering Task Force (IETF), RFC 1584, March 1994

[OMG] http://www.omg.org/

[PEN01] PENDERAKIS D., SHI S., VERMA D. and VALDVOGEL M., “ALMI: an Application Level Multicast Infrastructure”, *3rd USENIX Symposium on Internet Tehnol*ogies, San Francisco, CA, USA, Mar 2001

[PRI57] PRIM R. C., “Shortest Connection Netwotks and Some Generalizations”, Bell Sys. Tech. J., vol. 36, 1957, pp 1389-1401

[POS80] POSTEL J., “User Datagram Protocol (UDP”), RFC 768, August 1980

[POS81] POSTEL J., “Transmission Control Protocol”, DARPA Internet Program Protocol Specification, RFC 793, September 1981

[RAD01] RADHA KRISHNAN and BALAJI RAGHAVACHARI, “The Directed Minimum-Degree Spanning Tree Problem”, *Proceedings of the 21st Conference on Foundations of Software Technology and Theoretical Computer Science*, p.232-243, December 13-15, 2001

[RAM96] RAMANATHAN S., “An algorithm for Multicast Tree Generation in Networks with Asymetric Links”, *INFOCOM'96*. pp. 337-344

[RAV93] RAVI R., MARATHE M.V., RAVI S.S., ROSENKRANTZ D.J., HUNT H.B., “Many birds with one stone: Multi-objective approximation algorithms”, *Proc. of 25th Annual ACM STOCS* (1993), 438-447

[RIT84] Ritchie D.M., « A stream input-output system », *AT&T Bell Laboratories Technical Journal*, vol. 63, no. 8, 1984

[SARA] Simple Active Router-assistant Architecture, site web: http://enjambre.it.uc3m.es/~sara/

[SAL84] SALTER J.H., REED D.P, CLARK D.D., “End-to-end arguments in system design”, *ACM Transactions on Computer Systems. ACM*, 1984

[SCH93] SCHMIDT D. et al, “ADAPTIVE: A dynamically assembled protocol transformation, integration and evaluation environment”, *Concurrency: Practice/Experience,* vol. 5, no. 4, 1993

[SCH96] SCHULZRINNE H., CASNER S., FREDERICK R., JACOBSON V., "RTP: A Transport Protocol for Real-Time Applications", RFC 1889, January 1996

[SHE02] SHERLIA S., "Design of Overlay Networks for Internet Multicast", Doctoral Dissertation, Washington University in St. Louis, August 2002

[SMI92] SMITH W. D. and SHOR P. W., "Steiner tree problems", Algorithmica 7, pp. 329-332 (1992)

[SOL00] SOLEY R. and the OMG Staff Strategy Group, "Model Driven Architecture", Object Management Group, White Paper, Draft 3.2 – November 27, 2000

[STE00] STEWART R., XIE Q., MORNEAULT K., SHARP C., SCWARZBAUER H., TAYLOR T., RYTINA I., KALLA M., ZHANG L., PAXSON V., "Stream Control Transmission Protocol", RFC 2960, October 2000

[STU04] STUECKA ., "Validation of Communication Systems Using UML 2.0", White Paper, TELELOGIC documentation resources, April 2004, http://www.telelogic.com

[TAUG2] TAU Generation 2, site: http://www.telelogic.com/tau

[TELLO] Telelogic, site: http://www.telelogic.com

[UML20] http://www.uml.org

[VOG95] VOGEL A., KERHEVÉ B., VON BOCHMANN G. and GECSI J., "Distributed Multimédia and QoS: A survey", *IEEE Multimédia* Vol. 2, No. 2, P10-19, 1995

[VOS92] VOSS S., "Problems with Generalized Steiner Problems", Algorithmica 7, 333-335 (1992)

[WAI88] WAITZMAN D., PARTRIDGE C. and DEERING S., "Distance Vector Multicast Routing Protocol", Internet Engineering Task Force (IETF), RFC 1075, November 1988

[WAN00] WANG B. and HOU J.C., "Multicast Routing and its QoS extension: Problems, Algorithm, and protocols", *IEEE Networks*, Vol. 14, January 2000

[WET98] WETHERALL D., GUTTAG J. and TENNENHOUSE D., "ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols", *Proc. IEEE OPENARCH'98*, San Francisco, CA, April 1998.

[WIN92] WINTER P. and SMITH MACGREGOR J., "Path-distance heuristics for the Steiner problem in undirected networks", *Algorithmica* 7, 309-327 (1992)

[WRI00] WRIGHT L. K., MCCANNE S. and LEPREAU J., "A Reliable Multicast Webcast Protocol for Multimedia Collaboration and Caching", *Proceedings of the ACM Multimedia 2000 Conference*, Los Angeles, Calif. , November 2000

[YAM01] YAMAMOTO L., LEDUC G., "Autonomous Multicast Reflectors over Active Networks", *Symposium on Software mobility and Adaptive Behaviour*, *AISB'01 Convention,* York, UK, March 2001.

[YAN02] YAN S.Q., FALOUTSOS M. and BANERJEA A., "QoS-Aware multicast routing for the Internet: The design and evaluation of QoS MIC", *IEEE/ACM Trans. on Networking*, Feb. 2002, vol. 10 No.1, pp.54~66

[6WIND] 6Wind home site: http://www.6wind.com

[@IRS++] Architecture Intégrée de Réseaux et de Services Programmables Intégrant la Mobilité et le Multicast, RNRT @IRS++ Project, site: http://www-rp.lip6.fr/airs/

Author's Publications

Book Chapter

[GAR05] GARDUNO D., EXPOSITO E., DIAZ M., "Protocole de transport pour la diffusion multipoint multimédia à qualité de service", *Multicast multimédia sur Internet,* Chapitre 3, pp107-145, Edition Hermès, Février 2005

Conferences

[DIA01] DIAZ M., GARDUNO D., GAYRAUD T., OWEZARSKI S., "Multimedia multicast protocols based on multimedia models", *Invited paper. Multimedia Modeling Conference (MMM'01),* Amsterdam (Pays-Bas), 5-7 November 2001, pp.207-226

[EXP02] EXPOSITO E., SÉNAC P., GARDUNO D., DIAZ M., URUENA M., "Deploying new QoS aware transport services"; *International Workshop on Interactive Distributed Multimedia Systems and Protocols for Multimedia Systems (IDMS/PROMS'2002)*, Coimbra (Portugal), 26-29 Novembre 2002; and *Lecture Notes in Computer Science,* 2002(2515), pp.141-153, ISSN: 03029743

[EXP02-2] EXPOSITO E., SENAC P., GARDUNO D., DIAZ M., URUEÑA M., LARRABEITI D., "Déploiement de nouveaux services pour le transport de flux multimédia" ; *9ème Colloque Francophone sur l'Ingénierie des Protocoles (CFIP'2002)*, Montréal (Canada), pp.35-51. 27-30 Mai 2002

[GAR02] GARDUNO D., DIAZ M., GAYRAUD T., "An enhanced active traceroute", *International Symposium on Advanced Distributed Systems (ISADS'2002)*, Guadalajara (Mexique), 11-15 Novembre 2002; Proceedings of the International Symposium on Advanced Distributed Systems 2002, pp. 206-218, ISBN 970-27-0358-1

[GAR05-2] GARDUNO D., DIAZ M., GAYRAUD T., " Un Modèle d'Arbre Multipoint Orienté Qualité de Service Utilisateur", accepté dans le *11ème Colloque Francophone sur l'Ingénierie des Protocoles (CFIP'2005),* Bordeaux, France, 29 mars - 1er avril 2005

[URU02] URUEÑA M., LARRABEITI D., CALDERÓN M., AZCORRA A., KRISTENSEN J. E.,. KRISTENSEN L. K, EXPOSITO E., GARDUNO D., DIAZ M., "An Active Network Approach to Support Multimedia Relays", *International Workshop on Interactive Distributed Multimedia Systems and Protocols for Multimedia Systems (IDMS/PROMS'2002)*, Coimbra (Portugal), 26-29 Novembre 2002, pp. 353-364; and *Lecture Notes in Computer Science,* 2002(2515) pp. 353-364 , ISSN: 03029743

Project reports

[DIA01] DIAZ M., BAUDIN V., EXPOSITO E., GARDUNO D., HUTCHISON D., LARRABEITI D., MATHY L., OWEZARSKI S., PHAM-KHAC F., "Specification of the experimental platform", Deliverable Number 4.1.1, Project IST-1999-10 504 GCAP, July 7, 2001

[DIA02] DIAZ M., EXPOSITO E., GARDUNO D., SÉNAC P.,"Report and Assessment of Experiment 1 - Time-constrained multimedia server", Report Number D4.2.1, Project IST-1999-10 504 GCAP March 01, 2002

[DIA02-2] DIAZ M., BAUDIN V., EXPOSITO E., GARDUNO D., HUTCHISON D., LARRABEITI D., MATHY L., OWEZARSKI S., PHAM-KHAC F., "Report and Assessment of Experiment 2 - Videoconferencing and Multicast", Deliverable Number 4.3.1, Version 2, Project IST-1999-10 504 GCAP. Mars 22, 2002

[GAR01] GARDUNO D., OWEZARSKI S., DIAZ M., "GCAP: Experiment 2. Application messages", Rapport LAAS No01179, Project IST-1999-10 504 GCAP, Avril 2001, 31p.

[OWE01] OWEZARSKI S., GARDUNO D., GARCIA F., DIAZ M., "GCAP: Application level programming interface", Rapport LAAS No01180; Project IST-1999-10 504 GCAP, Avril 2001, 10p.

Internal reports

[EXP02-3] EXPOSITO E., SÉNAC P., GARDUNO D., DIAZ M., URUEÑA M., LARRABEITI D., "Deployment of new transport services for multimedia flows", Rapport LAAS N°02049, Février 2002, 17p