Université de Montréal

QoS Architecture and Monitoring for Videoconferencing Applications

Mathieu J. Poirier

Département D'informatique et de recherche opérationelle
Faculté des Arts et des Sciences

Mémoire présenté à
La Faculté des études supérieures
en vue de l'obtention du grade de
Maître (M.Sc) en informatique

Juillet 2000

Université de Montréal
Faculté des Études Supérieures

Ce mémoire est intitulé:

QoS Architecture and Monitoring for Videoconferencing Applications

Présenté par

Mathieu J. Poirier

A été évalué par un jury composé des personnes suivantes:

Esma Aimeur
Président-rapporteur

Rachida Dssouli
Directeur de recherche

Brigitte Kerherve
Membre du jury

Membre du jury

Mémoire accepté le: ________________

Remerciements:

J'aimerais remercier toute les personnes qui ont contribués à la réalisation de ce travail.

Dans un premier temps, j'exprime ma gratitude envers ma directrice de recherche, la professeur Rachida Dssouli, pour son support intellectuel et financier. Cette dernière m'a encouragé et supporté depuis le tout début. Le directeur de département de l'université de Montréal, le professeur Sang Nguyen, et des organismes subventionnaires comme le CRM et le CRSNG m'ont également offert un support financier. Je tiens aussi a souligner que les contributions du professeur Gérôme Canals furent indispensables pour la publication de notre article.

Mohamed Salem a grandement contribué à ce projet en formulant plusieurs conseils et opinions de qualité. Sa patience, sa dévotion et ses connaissances sont définitivement hors du commun.

J'ai une dette spéciale envers mes collègues de travail, Jean-Marc Ng Wing Keng et Yong Guo, qui ont grandement contribué à l'accomplissement de ce projet. Spécialement Jean-Marc, sur qui je me suis appuyé à plus d'une reprise.

Sincèrement votre, Mathieu.

Acknowledgments:

I would like to thank all the people that contributed to the accomplishment of this project.

First and foremost, I express my gratitude toward my research director, Ms. Rachida Dssouli, for her intellectual and financial support. She believed and supported me from the beginning. Sang Nguyen, head teacher of the computer science department at university of Montreal, along with other sponsoring bodies such as CRM and the CRSNG, also funded my work. It is important to outline that Gérôme Canals' efforts heavily contributed to the publication of our article.

Mohamed Salem has greatly contributed to this project through a number of opinion and advice that lighten several shadowed paths. His patience, devotion and knowledge are uncommon and to be cherished.

Finally, I have special debts toward my colleagues – Jean-Marc Ng Wing Keng and Yong Guo – who contributed to the realization of this project. Special thanks to the former for is support during harder times.

Sincerely yours, Mathieu.

À mes parent, sans qui rien de
ceci n'aurait été possible.

To my parents, without whom nothing
of this would have been possible.

Résumé:

Le déploiement massif de l'Internet s'accompagne d'un vaste développement des applications multimédias distribuées. Leurs domaines d'utilisation sont très vastes: télé-enseignement, systèmes de diffusion à la demande ou en direct, applications coopératives diverses. Traditionnellement installés sur des réseaux locaux ou RNIS (Réseaux Numériques à Intégration de Services), les systèmes de vidéoconférences étaient jusqu'ici construits principalement sur des protocoles propriétaires. Leur déploiement sur Internet impose l'utilisation de protocoles plus standardisés qui permettent l'adaptation aux contraintes du réseau et un certain niveau d'interopérabilité.

La principale caractéristique des applications multimédia est l'introduction et la manipulation de médias continus tels l'audio, la vidéo et le transfert d'information sensible au temps. Leur défi est de satisfaire aux exigences des usagers en fonction des possibilités du réseau utilisé. Les solutions proposées, et celles qui restent a venir, tournent toutes autour du concept de « Qualité de Service » (QdS). Ce dernier désigne un ensemble de propriétés, perceptibles par l'utilisateur, qui caractérisent la performance du service qui lui est offert. Actuellement, l'Internet ne peut supporter de telles garanties et offre un service qualifié de «*best effort*». La gestion des paquets de données est prise en charge de façon simple mais ne garantie ni la livraison ni la ponctualité.

Pour satisfaire les contraintes temporelles induites par la manipulation de données continues, un ensemble de fonctionnalités de gestion de la QdS est nécessaire. Beaucoup de recherches en ce sens ont focalisé sur *la réservation de ressources* dans des réseaux à circuit virtuel tel qu'ATM. Le modèle de commutation par paquets qui est à la base des réseaux IP rend l'application de la réservation de ressources impossible. De nombreuses techniques basées sur la différentiation de paquets et de classes de services ont cependant été proposées afin de d'introduire une certaine QdS dans l'architecture d'Internet.

Dans ce mémoire, nous suggérons une architecture qui intègre des mécanismes de contrôle et de gestion de la QdS. L'accent est mis sur les stratégies d'adaptation qui permettent à l'application de réagir à des fluctuations de services du réseau. Nous introduisons le concept de réseaux multiples, où une machine client et une machine serveur sont reliées par plusieurs réseaux offrant une QdS différente. Un prototype de support à une application a été développé en tenant compte des mesures de QdS. Le prototype est conforme aux standards afin d'être compatible et interopérable avec d'autres applications suivant ces mêmes normes. Nos travaux s'appuient sur le standard H.323 qui définit des spécifications pour la communication multimédias sur réseaux à commutation par paquets tel Internet.

Comme on le mentionne dans le paragraphe précédent, nous proposons de travailler sur trois plans: l'amélioration de la définition de la QdS et le support qu'elle requiert, l'interopérabilité, et finalement la polyvalence de l'application. Voici un bref aperçu des solutions et stratégies adoptées dans chacun de ces aspects.

Le concept de QdS et la façon de l'utiliser sont à la base de notre travail. Des recherches approfondies nous ont permis de comprendre que les paramètres de QdS implantés dans les rares applications qui les supportent sont souvent trop près des mesures de réseau. Dans ces circonstances, un certain niveau d'expertise est requis par l'utilisateur. Afin d'en faciliter la compréhension et la manipulation, nous introduisons des classes de services qualitatives plutôt que quantitatives. Ces dernières sont ensuite transposées en mesures quantifiables afin d'être prises en charge par un réseau. Toujours en fonction des préférences de l'usager, différents paramètres de réseau (délais, intervalles d'arrivé, nombre de paquets perdus) peuvent être pris en compte.

Ces derniers sont par la suite soumis à un algorithme d'adaptation qui ajustera les caractéristiques d'un flot binaire en fonction des fluctuations de service. L'algorithme est capable de travailler avec un des trois paramètres spécifiés plus haut. Les niveaux de tolérence et les actions à prendre dans certaines situations critiques sont entièrement contrôlés par l'usager. Par exemple, ce dernier peut demander de favoriser les connexions audio au dépend des connexions vidéo, avec une vitesse de transmission minimale de 20 Kb/sec pour l'audio et 50 Kb/sec pour la vidéo. Chaque connexion est traitée de façon individuelle, ce qui implique que chacune a sa propre spécification de QdS et que ces spécifications ne sont pas inter-reliées.

Afin de faciliter la gestion des connexions et de gérer l'algorithme de dégradation, une série d'objets offrant une abstraction de haut niveau (API – Application Programming Interface) à été développée. En plus d'encapsuler plusieurs processus de bas niveaux, ces objets permettent de faciliter la manipulation du système et de réduire le nombre de tâches d'implantation.

Puisque notre système est destiné à la formation de personnel à distance, il est important qu'il puisse interagir avec d'autre applications qui pourraient éventuellement avoir été conçues par d'autres équipes. Puisque nous travaillons dans un environnement distribué, il est également préférable que l'architecture suive les directives d'implantation standards afin d'en améliorer la polyvalence.

Pour ce faire, nous appuyons nos efforts sur deux spécifications standards qui servent actuellement de plan d'implantation dans plusieurs produits commerciaux. L'une d'elle spécifie les protocoles RTP et RTCP tandis que l'autre se concentre sur l'architecture H.323.

RTP est un protocole de couche application qui encapsule les données multimédia en fonction de spécifications précises. De plus, son en-tête inclut plusieurs champs d'information généralement nécessaire pour la reconstruction des signaux. Des numéros de séquence, des estampilles temporelles et l'identification du mode de codage en sont des exemples. RTCP est le protocole de contrôle qui introduit les mesures de qualité de services offertes par le réseau. La manipulation de ces mesures donne de l'information sur les performances de transfert qui autrement ne figurent pas dans les services offerts par couche IP.

H.323 est définitivement la plus complète des spécifications en ce qui concerne l'implantation de système multimédia sur réseau à commutation – IP, Ethernet, cell relay, ... Cette spécification est la fondation de notre travail. En s'y conformant, il nous est possible d'espérer que notre application aura certaines fonctionnalités communes avec des systèmes développés par d'autres équipes. H.323 indique les normes à suivre et les types d'encodages à utiliser pour optimiser l'acheminement de contenu médiatique. Il utilise également les protocoles RTP/RTCP présentés ci-haut pour l'encapsulation et l'acheminement des données. Ce standard est sans contredit voué à un avenir très florissant. En effet, plusieurs compagnies l'implantent actuellement dans une vaste gamme de produits allant de l'ordinateur de bureau au téléphone numérique.

En troisième lieu, nous abordons le concept de polyvalence en modifiant le modèle de couche réseau actuel en ajoutant la possibilité de communiquer avec plusieurs interfaces réseau simultanément. Avec ce nouvel apport, il est possible pour un usager d'acheminer un flot de voix sur réseau ISDN, un flot de données sur ATM et d'effectuer un transfert de fichier sur IP parallèlement. Toute la gestion des différentes couches transport est prise en charge par l'architecture et les méthodes de gestion de QdS mentionnées plus haut.

Afin de concrétiser le concept, nous dotons notre architecture d'une couche transport qui identifie le réseau sur lequel est destiné un paquet en fonction des spécifications préalablement établie par l'usager. Afin de prioriser certains types de médias, nous ajoutons trois files d'attentes pour chaque interface réseau: audio, vidéo et données. De ce fait, l'algorithme de séquencement peut sélectionner les données à acheminer en fonction de leur importance pour l'usager et leurs caractéristiques temporelles. De plus, nous permettons à l'architecture de permuter les réseaux de support lorsqu'un de ceux-ci n'est pas en mesure de supporter les paramètres demandés. Par exemple, si le système détecte une vitesse de transmission trop faible sur IP, il peut décider d'acheminer l'information sur ATM. Le changement de réseau s'effectue alors sans interruption de service et l'usager est informé de l'événement par un des panneaux de contrôle.

Un prototype concrétisant l'architecture présentée ci-dessus a été réalisé et expérimenté afin de valider notre proposition. Le prototype implante de manière presque complète cette architecture.

Du côté du serveur, l'expérimentation s'est faite sans codecs (pour des raisons de coût). Le générateur de flux utilisé ne produit pas un flux réel mais simule des données. Les niveaux de services accessibles à l'usager ont également été simplifiés. Le prototype actuel travaille avec 5 classes de service qui gouvernent la vitesse de transmission des flots de données. D'autres paramètres comme le nombre de paquets perdus ou l'intervalle de temps entre la reception des paquets pourraient également être utilisés, mais n'ont pas été implantés dans le protoype faute de temps.

Du côté client, une interface utilisateur à été installée au dessus de l'API de QdS. Cette interface présente l'évolution dynamique des paramètres réseaux tel que rapportés par les paquets RTCP.

Enfin, des protocoles de gestion des connexions ainsi que des modules de support ont été introduis pour simplifier l'interaction avec les protocoles H.225 et H.245. Ces derniers étaient nécessaires pour alléger la charge de développement.

Nos simulations ont été obtenues à partir de tests de performance effectués à différentes périodes de la journée dans le laboratoire de téléinformatique de l'université de Montréal. Les équipements de support sont des ordinateurs de type PC fonctionnant sous la tutelle du système d'exploitation windowsNT 4.0, des processeurs 433 MHZ Celeron d'Intel, et des cartes réseaux ethernets à 10Mbs. Le réseau interne du laboratoire est organisé en étoile avec des liens de fibres optiques à 100Mbs et un *hub* cental qui travaille généralement à 75% de ses capacités.

Bien que cet environnement de test ne reflète pas exactement les conditions d'engorgement d'un Internet ouvert, nous sommes d'avis que l'hétérogénéité des ordinateurs et des applications qui l'utilisent sont près de la réalité. En effet, ce même réseau est utilisé pour connecter des ordinateurs PC, Mac, Sun et HP. Les systèmes d'exploitations et les services demandés varient grandement d'une plate-forme à l'autre, ce qui introduit différentes formes de dégradations de services allant d'engorgements extrèmes à une bande passante maximale.

La réalisation de ce projet présente indéniablement plusieurs défits de tailles à relever. Dans un premier temps, nous avons dû définir le concept de qualité de service ainsi que les façons de l'appliquer et ses impacts sur les applications distribuées. La définition et l'étendue des spécifications des systèmes collaboratifs ont également posés plusieurs problèmes. Ces derniers sont en effet très complexes à comprendre et à implanter puisque plusieurs outils tel que planche à dessins, partage d'applications et soutient de média font parties de ses caractéristiques.

Le fait de travailler avec des spécifications standards a considérablement alourdi la tâche. À cause de contraintes temporelles, il nous a été impossible d'implanter tout les services en fonction des spécifications. Plusieurs protocoles ont été simplifiés et modifiés pour exhiber uniquement les caractéristiques nécessaires. Nous sommes conscient que ces altérations causeraient des problèmes d'interopérabilité dans l'optique un déploiement sérieux.

Finalement, un nombre important de considérations sont toujours requises afin d'amener le prototype à un produit mature. Notamment au niveau de l'acquisition, la compression, la paquetisation, la synchronisation et le rendu des données multimédia. D'autres défis se situent au niveau de la représentation, la réservation et l'acquisition de ressources dans les systèmes d'exploitation et les équipements de réseautique afin de concrétiser le concept du chemin bout-en-bout.

La structure de l'ouvrage se présente comme suit:

Après une brève introduction décrivant le contexte de notre travail, nos motivations et les stratégies proposées, le deuxième chapitre traite des caractéristiques propres aux applications multimédia. Plus précisément, nous tentons de classifer les différentes variantes en plus d'en définir les caractéristiques propres.

En troisième lieu, nous passons en revue des concepts classiques reliés à la définition, le support et la gestion de la qualité de services. Sur la même lançée, certaines méthodes permettant de concrétiser les concepts précedents sont présentées. En plus d'avoir fait l'objet de nombreux ouvrages, les algorithmes choisit sont couramment implantés dans les équipements de télécommunication actuels.

Le chapiter 4 traite des travaux déjà réalisés dans le domaine de la qualité de service et des supports qu'elle requiert. Nous présentons donc différentes architectures développées par des groupes de recherche reconnus pour leurs expertises et savoir-faire dans ce champ d'intérêts.

Nous voyons par la suite (chapitre 5) certains protocoles applicatifs ayant été définis pour le support et le contrôle d'information multimédia. Plus précisément, les protocoles RTP, RTCP et RTSP sont expliqués. Travaillant en relation étroite, ils offrent plusieurs fonctionalités qui complètent et enrichissent les services actuels.

Le sixième volet de l'ouvrage engloble l'ensemble du matériel abordé jusqu'ici en décrivant le standard H.323. Celui-ci est un plan détaillé qui spécifie le matériel et les protocoles devant interagir afin de rendre possible le support d'audio et de video dans des environnements qui n'offrent pas de qualité de service – typiquement Internet. Depuis son adoption en 1998, H.323 s'est imposé comme référence dans le développement de systèmes interopérables.

Notre architecture, décrite dans le chapitre 7, est également basée sur le standard H.323. Nous ajoutons à la norme plusieurs modules qui introduisent de nouveaux services. Les nouvelles fonctionalités se situent au niveau de la définition, la déclaration et le traitement des paramètres de qualité de service. Nous ajoutons également des moyens permettant de mesurer le niveau de service donné par un réseau ainsi que des mécanismes pour s'y adapter. Afin de maximiser les performances de transfert, la couche "transport" de notre modèle fût enrichie d'interfaces capables d'interagir avec plusieurs modes de commutation simultanément.

La réalisation d'un prototype est décrite dans le chapitre 8. On y voit comment les solutions proposées travaillent, l'efficacité des algorithmes d'adaptation et certaines interfaces de manipulation. Malgré plusieurs embûches d'ordre technique, la plupart des défis reliés à la gestion de la QdS de bas niveau ont été réalisés avec succès.

Nous terminons le travail en donnant une liste de nos réalisations ainsi que l'apport qui s'en suit. Des indications générales quant à l'avenir du projet et des idées de travail futur sont également mentionnées.

Abstract:

Unlike other technologies such as ATM or ISDN, IP networks don't offer provisioning mechanisms to meet real-time requirements introduced by multimedia content.

Today's applications need guaranteed end-to-end service classes – namely QoS – capable of coping with time constraints required by delayed sensitive data streaming. In network equipment, the above services are modelled by preferential packet treatment to enhance rate forwarding and reduce packet queueing to a minimum. As for end-systems, they translate in a set of mechanisms to declare, reserve and control quality parameters.

In this work, QoS is motivated by the need of distant learning and implies the distribution of time sensitive data on networks that cannot provide information concerning their status and offered services. More precisely, we give details about functionalities added to the H.323 standard to support user defined QoS specifications.

The intention of designing such architecture is to establish a set of quality of service configurable interfaces that formalize quality of service in the end-system and network, providing a framework for the integration of QoS control and management mechanisms. Finally, we add our personal flavor by enabling the stack with several network interfaces capable of seamlessly dealing with a variety of network layer technologies.

Key words: Quality of services, real-time requirements, media streaming, QoS Architecture.

Table of Content:

List of Figures:

Abbreviation:

AAL:	ATM Application Layer.
AAL5:	ATM Application Layer version 5.
AF:	Assured Forwarding.
API:	Application Programming Interface.
A/V:	Audio and Video.
ATM:	Asynchronous Transfer Mode.
CODEC:	Coder / Decoder.
DCT:	Discrete Cosine Transform.
DiffServ:	Differentiated Services Architecture.
DMA:	Distributed Multlimedia Application.
DMS-CC:	Digital Media Storage – Command and Control.
DS:	Differentiated Services.
EF:	Expidited Forwarding.
FIFO:	First in, First out.
HDTV:	High Definition TeleVision.
HTTP:	HyperText Transfer Protocol.
IETF:	Internet Engineering Task Force.
InstServ:	Integrated Services Architecture.
IP:	Internet Protocol.
IPv4:	Internet Protocol version 4.
IPv6:	Internet Protocol version 6.
ISDN:	Integrated Service Digital Network.
ISP:	Internet Service Provider.
ITU:	International Teleconferencing Union.
LAN:	Local Area Network.
LDAP:	Lightweigh Directory Access Protocol.
LDP:	Label Distribution Protocol.
MCU:	Multipoint Control Unit.
MM:	Multimedia.

MPLS:	Multi-Protocol Label Switching.
MSC:	Message Sequential Chart.
NTP:	Network Time Protocol.
OS:	Operating System.
OSI:	Open System Interconnection.
PDU:	Protocol Data Unit.
PHB:	Per-Hop-Behavior.
POTS:	Plain Old Telephone System.
PVC:	Permanent Virtual Circuit.
QoS:	Quality of Service.
RAS:	Registration/Admission/Status.
RFC:	Request For Comments.
RSVP:	ReSerVation Protocol.
RTCP:	Real-Time Control Protocol.
RTP:	Real-Time Transport Protocol.
RTSP:	Real-Time Streaming Protocol.
RTT:	Round Trip Time.
SLA:	Service Level Agreement.
ST-II:	Steam Transport Protocol version 2.
TCP:	Transport Control Protocol.
ToS:	Type of Sercice field of an IP packet.
UDP:	User Datagram Protocol.
VC:	Virtual Circuit.
VPN:	Virtual Network Path.
WFQ:	Weighted Fair Queueing.

1 – Introduction:

1.1 – Overview:

For more than two decades now, computers have exchanged different kinds of information by making use of network infrastructure. In the early beginning, researchers scattered among research institutes were the first beneficiaries. Since then, many efforts have been made to open computer networking services to a broader range of users. Ongoing improvement of the available bandwith, the introduction of efficient transfer protocols and the stumbling of computer prices are all factors that contributed to exponentially increase the amount of interconnected people.

Surprisingly enough, one of the oldest network protocol not only survived the evolution wave but also established itself as the foundation of the most popular network switching scheme in use today. IP, the Internet Protocol, is the enabling technology governing the Internet, a giant cloud of connected network all speaking the same common language as defined by the IP standard protocol.

Throughout the years, several factors have kept IP on top of networking technologies. Among them, its flexibility, reliability and efficiency in transporting documents among distributed sites were particularly appreciated. In fact, IP was among the first protocol to be defined and since it provided so much functionality, people simply continued to use it.

With the advent of available bandwidth and network connectivity, more and more users became acquainted with network services. Nowadays, the Internet is a communication tool used for an infinite number of purposes. Entertainment, advertisement and distributed work are valid examples. The services demanded by the ever-growing community of users has not only increased, but also changed. Indeed, it is now familiar to encounter live media broadcast, animation and interactive environment on the Internet.

1.2 – Problem Description:

The nature of the above-mentioned services varies significantly from the former and more conventional static data exchanges. In fact, they pause severe network constraints that were initially not part of the requirements and information transfers suffer therefore from various transition hazards.

When static content is sent from one host to another, it is desirable for data to reach its destination as soon as possible. Nevertheless, moderately long end-to-end delays, up to tens of seconds, are often tolerated. On the other hand, audio and video content are highly sensitive to delay introduced by the best-effort service offered by IP's switching scheme. They are also tolerant to loss; occasional packet dropping causes minor glitches in the audio/video playback, and can often be partially or fully concealed by interpolation algorithms. Thus, in terms of service requirements, streaming applications are the opposite of static-content applications: multimedia applications are delay sensitive and loss tolerant whereas static-content applications are delay tolerant and loss intolerant [53].

In today's world of network interconnection, the paradigm of 'Quality of Service' (QoS) is becoming the ultimate goal. QoS is the enabling factor for time sensitive information distribution – audio, video, and real-time data – over IP networks. This notion had been used with various meanings, though we take it in the context of multimedia application where the following definition can be applied: " The quality of service represents the set of those quantitative and qualitative characteristics of a distributed multimedia system necessary to achieve the required functionality of an application" [91].

Defining QoS is a challenging task since its characteristics and the pertaining services significantly change with the context in which it is used. Moreover, understanding and manipulating the wide range of description parameters is often too difficult for unacquainted users. The latter are usually more interested in the services yielded by an application and thus are less concerned with complex environment settings.

Incorporating the concept of QoS in actual IP networks basically means that core-switching equipment must be capable of differentiating data packets to provide faster treatments. Moreover, streaming applications must request and reserve sufficient amounts of resources in the system to guarantee that real-time processes will be serviced in accordance with their time requirements. The latter two form an end-to-end guaranteed path, from application-to-application, that is capable of assuring flow service from the remoter server, in the network, to the intended client(s). This end-to-end path must also follow a common representation to enable equipment and applications developed by different vendors to efficiently communicate and work together.

Finally, sustaining realistic QoS is hardly achievable with a single network technology. Indeed, each switching scheme (ATM, IP, cell relay, etc.) offers different service possibilities that may not be found in other networks. Such possibilities are affected by the underlying network technology, the type of connection (connection oriented or not), the available bandwith and the operating costs. Nowadays, many new technologies are available to users at affordable prices, but little – or nothing – has been done to incorporate them in a single and easy-to-use platform.

The above paragraphs depict the problems addressed in this thesis: 1) the difficult task of defining, specifying and manipulating QoS parameters, 2) a lack of standardization to properly specify QoS requirements among vendors, and 3) the absence of support for multiple network interfaces to take full advantage of available network technologies.

1.3 – Goals and Motivations:

Researchers around the world envisioned the situation and began to think about ways to enable realistic media streaming over IP. To this date, considerable achievements have been made and several prototypes proved the concept feasible.

Nevertheless, the suggested solutions are generally incomplete in a number of ways. Most are tied to restrictive evolution contexts in which bold assumptions are made – some not even applicable to an IP based network. They lack mechanisms to define, control and support QoS parameters and sometimes rely on network protocols to guarantee their services. Finally, an overall framework is needed to develop architectures to build upon and offer media streaming among different vendors.

In this work, QoS is motivated by the need of distant learning and implies the distribution of time sensitive data for networks that cannot provide information concerning their status and offered services. The usual network that refers to such a characteristic is governed by the Internet protocol. Our goal is to produce a platform capable of sustaining real time requirements for distant learning medical applications.

To solve the constraints added by heavy-streamed media on IP and promote tool interoperability, in 1998 the ITU approved the ITU approved the H.323 protocol stack – an open framework that guides implementers in their design choices. The goal of our work is to produce an architecture that is both compliant to H.323 and capable of offering user specific QoS. The intention of defining such architecture is to establish a set of quality of service configurable interfaces that formalize quality of service in the end-system and network, providing a framework for the integration of QoS control and management. Finally, we add our personal flavor by enabling the stack with several network interfaces capable of seamlessly dealing with a variety of network layer technologies.

1.4 – Organization of the Work:

To correctly address the challenges related to the implementation of QoS in multimedia applications, a myriad of topics must be reviewed and understood. In this context, the actual document is organized as follows.

Our quest starts in chapter 2 where multimedia applications are presented. More precisely, they are classified in accordance with their characteristics and the context in which they work. Since they deal with highly sensitive content (audio and video), a list of requirements guaranteeing optimum results is presented along with a description of how they affect ongoing presentations.

In chapter 3, general mandatory concepts believed to enable QoS provisioning in IP networks are listed. Such concepts are commonly encountered in work involving QoS implementation and have proven to be useful starters in many realizations. Directly after, we introduce different implementation techniques that reproduce the behaviors described in the above. Basically, they are scheduling algorithms for core network equipment and traffic modeling for flow shaping in end-system platforms. It closes with QoS management activities that provide interfaces for administrating flows and sustaining requirements throughout the data path.

Chapter 4 concentrates on end-systems. End-systems are usually called distributed multimedia applications since they support media streaming and work in heterogeneous environments connected by multiple network technologies. The chapter describes the essence of existing multimedia applications developed by leading universities renowned for their work in the support of time sensitive flows over network links. The most successful projects are presented along with an assessment of their pros, cons and feasibility.

Chapter 5 presents a set of protocol enabling standard distribution and manipulation of media streams. More precisely, RTP (Real-time Transport Protocol) provides a common way to encapsulate media information in data packets to be sent on the network. Its companion control protocol, RTCP (Real-time Transport Control Protocol), allows application level monitoring of the data delivery, a feature not provided by IP. It also conveys minimal control and identification functionality. Thirdly, the Real-time Streaming Protocol (RTSP) is briefly approached. The latter serves to manipulate media stream with video-like commands, e.g. play/pause.

Following (chapter 6) is the presentation of the H.323 protocol umbrella. It is a flexible yet powerful specification covering several aspects of media distribution in packet based network lacking the support of service specification – typically IP, but other technologies such as Ethernet can apply. It addresses problems pertaining to time constraints and network deficiencies by offering a mean for client application to select parameters that are best suited for their environment. It also delivers a standard set of technology requirements (codec, protocol, signaling) that insures common understanding among all participants.

Chapter 7 and 8 are respectively concerned with the design and implementation of a QoS aware protocol stack suitable for the support of media streaming for distant learning applications. Our design choices are explained along with the assumptions and tradeoffs that were made. Despite many conception hazards, a working prototype has been realized, proving that our solutions are viable. We conclude by looking back at our work, assessing achievements and problems encountered. Finally we address future work required to bring the prototype to full implementation.

1.5 – Author's Contributions:

The project described in this thesis has been implemented by a joint effort from the author and two other students – Jean-Marc Ng Wing Keng and Yong Guo. Although each participant has contributed to all design aspects of the work, Jean-Marc Ng Wing Keng and Yong Guo were respectively concerned with implementing the multi-transport layer and the QoS definition and management. The author's contributions mostly related to handling or imitating H.323 services and are specifically outlined in the following list. They correlate to aforementioned goals by providing basic connection and protocol support to higher level applications and other services needed by different modules of the architecture.

- Design and implementation of a client module capable of working with the server prototype.

- Implementation, generation and management of RTP/RTCP simulation packets whose base mnemonics has been taken from a third-party library.
- Production of QoS information based on the reception of RTCP packets.
- Design and implementation of a set of protocols imitating connection-specific H.323 protocols.
- Design and implementation of a global framework into which modules from other collaborators were inserted.
- Design and implementation of a messaging API allowing reliable, asynchronous message exchanges between architecture components.
- Design of the state machine allowing the introduction, manipulation and departure of users in multimedia session.
- All other H.323-related aspects such as protocol definition, media streaming and connections specification, establishment and management.

2 – Distributed Multimedia Applications:

All the concepts relating to QoS have one thing in common: They all pertain to specific applications called *Multimedia Applications*. Nowadays, most are connected by network infrastructure and are thus referred to as *Distributed Multimedia Applications* (DMA). The characteristics of these differ in many ways from classical applications handling static data such as word processors, internet browsers and presentation tools. In fact, they are so special that their design and the environment in which they execute must be carefully planned to ensure successful results.

The specific issues to consider when dealing with MM applications originate from the type of content they usually manipulate, that is, audio and video. The latter are both particularly sensitive to processing delays in the end application and network links, while being relatively tolerant to packet loss. This is strikingly different from legacy applications in terms of requirements: MM tools are delay sensitive and tolerant to loss whereas static content applications are unaffected by delay but highly disturbed by loss.

2.1 – Classifying Multimedia Applications:

Most MM applications can be classified in three groups that broadly encompass all capabilities. These classes are presented below with a description and some implementation examples [37,39,53].

2.1.1 – Broadcasting Applications:

As implied by the name, this type of application is similar to regular broadcast of radio or video with the exception that signals are delivered over the Internet instead of airwaves or copper wires. Similarly to television and radio, users can browse between channels carrying unicast or multicast transmissions.

The difficulty in building broadcasting applications relates to the important throughput and minimum delay required. This is especially true with live broadcast where acquisition, compression, delivery and rendering are made in real-time. When acquired from a permanent storage, care must be taken to ensure that all equipment on the data path is capable of sustaining the required time constraints. This is particularly difficult since traditional file systems are characterized by slow seek time and network links most likely congested. Nevertheless, successful implementations of architectures and prototypes had proven the concept feasible. Such examples can be found at Berkeley [1, 60,65], Lancaster [59] and Montreal [49].

2.1.2 – Media-On-Demand Applications:

Those can be seen as a subset of the broadcasting applications depicted above. In the same way, they are used to retrieve media files from a remote host. The distinction lies in the following aspects: 1) they do not support live broadcast, 2) unless it is multicast enabled, they don't support large audience screening and 3) they are mostly used for working or learning.

With this class of application, a client requests on-demand media files stored on servers and after a small delay (within ten seconds), the playback begins on the client's machine while it continues to arrive. Interactivity is also a common feature that allows one to temporally alter media states, e.g., play/pause/resume.

Since they don't have to deal with hard real-time requirements, media-on-demand applications impose less stringent demands in terms of delay and jitters and are therefore easier to build. As such, several commercial applications are available including RealPlayer from RealNetworks [71], NetShow from Microsoft [61] and Internet Wave from Vocaltec [90].

2.1.3 – Cooperative Work Tools:

Cooperative work tools are by far the most complex and difficult applications to build. This is caused by the amount of requirements and the functionality needed to allow several people to collaborate in real-time over network links. Examples of such applications include tele-conferencing, tele-education, tele-commuting, and remote medical systems [37]. These kind of distributed meetings also permit one to share documents, manage sessions, do floor control, support audio/video streams and synchronize participants to ensure peaceful workflow.

Most cooperative work tools gather both the functionality of broadcasting and media-on-demand applications, in addition to other utilities such as whiteboards, shared presentation and remote appliance control. The available tools are not usually capable of fulfilling all requirements and custom parts and modules must therefore be added.

The most popular cooperative application on the market today is definitively NetMeeting by Microsoft [61]. This popularity stems from the fact that it's very functional and relatively simple to use, although not ready for extensive traffic and still under Microsoft's ruling servers. Other applications were produced in the academic and scientific communities. Some examples of successful projects are the JVTOS (Joint Viewing and Tele-Operation Service) [27,36], the Upper Atmospheric Research Collaboratory [68] and the Collaborative Remote Observing at W.M Keck Observatory [50].

2.2 – Multimedia Application Requirements:

In this sub-section, we take a look at some general requirements needed by multimedia applications. Here, we make no discrimination about the class to which an application belongs since the following are believed to be relevant to all three classes. Even if the discussion tends to be generic, we focus on requirements that can sustain media *playback*.

As stressed before, the handling of audio, video and real-time data transfer is significantly different from classic data content and affects both the communication links and the end-points in terms of throughput, time constraints, service commitments and group communication [3,20,75].

2.2.1 – High Throughput:

Handling MM files introduces an incredible amount of data to be supported and managed by underlying network systems (Table 2.1). The bandwidth needed for video transmission ranges from $p \times 64$ Kbit/s – with p taking the value of 1 to 30 for ISDN channels – to 1.2 Gbits for High Definition TV (HDTV) [56,88]. Data rates are usually reduced by compression algorithms bringing less flexibility, increased image degradation and augmented latency. Even with the actual compression standards, audio and video streaming still take a heavy toll on networks and platforms. Unless high-speed networks and real-time operating systems are available, realistic media streaming is not feasible.

Media	Avg. bit rate
Uncompressed CD-quality audio	100-200 Kbit/s
Uncompressed standard-quality video	140 Mbit/s
Uncompressed high definition HDTV	1.2 Gbit/s
Compressed HDTV	128 Mbit/s
MPEG video	≅ 2Mbit/s

Table 2.1: Average data rates of some media [35].

Today's MM applications working over IP networks typically use the H.261 and H.263 codecs, whose lowest bit rates are respectively 64 Kbit/s and 20 Kbit/s [21].

2.2.2 – Time Constraints:

Media streams are closely related to temporal dimensions resulting in a set of synchronization requirement [39] mapped in three categories: stream synchronization, event synchronization and group synchronization.

Stream synchronization:

Stream synchronization deals with delays and jitters introduced between the segmentation of the signal at the origin and its reconstruction on the client's machine. In agreement with Hafid [37,39] and Kurose [53], two kinds of stream synchronization exist (Figure 2.2):

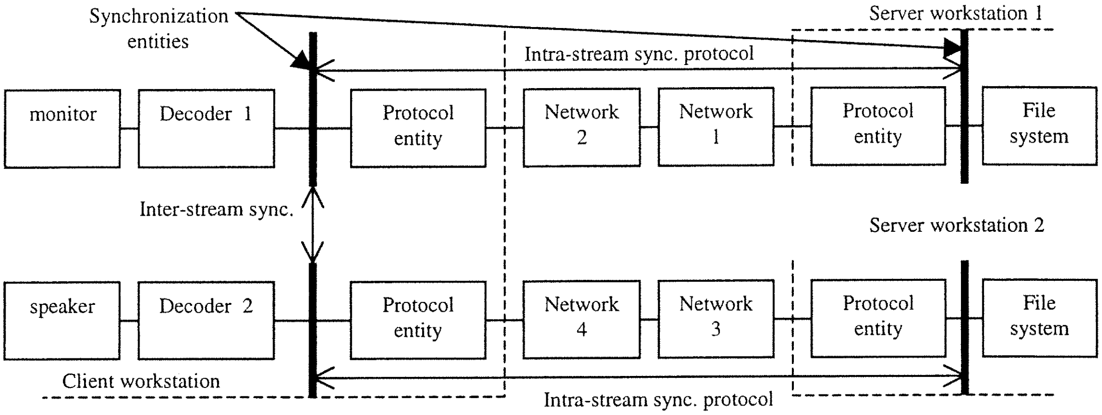


Figure 2.2: Inter/Intra stream sync. [52].

- Intra-Stream Synchronization: This is to guarantee an upper limit on the acceptable delay that any data will experience. If this upper bound is passed beyond a set point, the information becomes useless. Each step between the media generation on the server's machine and the presentation at the client side contributes to a fraction of the total delay.
- Inter-Stream Synchronization: Sometimes related streams are sent over networks in separate flows. We assume that prior to their separation, streams were synchronized and thus need to be re-synchronized when played at the other end.

Event Synchronization:

Many kinds of events can occur in a MM application. Depending on their nature, they inform the system that the network state had changed, that someone joined or left a group, that the state of a stream had been modified, etc. They induce severe constraints since each time such events happen, proper actions must be taken while still guaranteeing ongoing services.

Group Synchronization:

Group synchronization deals with the concept that everyone participating in a session must see the same thing. All members of a group must therefore have the same view of shared windows at virtually the same time. This is especially important for applications concerned with interactions between several users who may receive the same media stream at the same moment. Group synchronization must be supported by an important range of applications such as teleconferencing systems and collaboration tools [37].

2.2.3 – Service Commitments:

It was said earlier that MM streams have service requirements well above the more conventional static data. This means that all equipment along the data path, server and client machines included, must be capable of reserving and sustaining a pre-established service to avoid delays, jitters and packet loss.

The problem is to know exactly, at all times, what resources are committed and how much can be given to accommodate a maximum amount of clients with as many guarantees as possible. Resource allocation is difficult to achieve because it requires the handling of a call request that states future service requirements. Even when such service is available, it is still problematic to keep track of all accepted connections since memory and physical storage are limited.

2.2.4 – Group Communication:

Two things must be considered when talking about group communication. First, how participants communicate when working in the same group and secondly, how media stream reach the intended users. In the former case, different network topologies can be established to allow participants to talk to any of the other peers [83]. High point-to-point or multipoint connections must therefore be created to service the $(n! \times (n - 1)!)\ 2!$ possible links – where n is the number of participants. For very large groups, point-to-point link management becomes too heavy and other group communication mechanisms like sub-groups and multicast addresses must be used.

In the second case (reaching participants), stream broadcasting with a multicast enabled network is more efficient than traditional unicast – but most people are still connected by the fourth version of IP, which doesn't offer such a feature. Therefore, we still face a scalability problem in point-to-point connections for larger applications. Multicast networks like IPv6 have been designed with this exact idea in mind [30, 51]. Instead of establishing one link per media, groups are maintained in network routers responsible for the replication of media packets. Servers are thus relieved from the expensive duplication overhead.

Although convenient from a scalability point of view, multicast networks also bring their share of burdens. This is the case when clients having different processing powers share the same stream. In this situation, multilayered media streams must be used to accommodate both fast and slow machines [62, 69, 82].

2.3 – Summary:

The past section gave an insight about distributed multimedia applications and their characteristics. We began by classifying them into three broad classes generally encompassing the available implementations. Applications belonging to the third class, the cooperative work tools, are particularly difficult to implement due to the enormous amount of specifications and interacting technologies. Nevertheless, successful implementations had proved the concept feasible and brought interesting results to their users.

Secondly, we looked at MM application requirements and outlined the fact that they significantly differ from conventional designs. High throughput, restrictive time constraints, service commitment and group communication are all aspects to address when designing streaming tools.

The next section tackles the problem of differentiating time sensitive from non-time sensitive packets in core network equipment. Favouring media packets is especially important since it reduces delays and latency in the end-to-end path. To this date, several propositions have been envisioned to replace IP's best-effort model that is not capable of such differentiation.

3 – QoS Principles for Packet Switched Network:

The support of QoS classes is crucial to differentiate audio and video packets from non-real time packets in network equipment. It is well known and understood that multimedia applications need QoS that goes beyond the original best effort service offered by the Internet. But it is very difficult to provide QoS on IP networks since the original architecture was not designed to support real-time constraints. The following constitute the basic knowledge required to understand and implement packet differentiation (hence, QoS) in IP. This introductory material is currently used by governing QoS standards and will be referred to in later sections of this work.

The concepts explained in this section are widely used in telecommunication networks as means to regulate packet forwarding. IntServ [9,10,96,97], DiffServ [6,41,48] and MPLS [2,14,43,72] are commonly known as leading switching scheme enabling packet differentiation, therefore implementing different classes of service.

3.1 – Four Principles for Providing QoS Support:

QoS is not easy to sustain in IP networks. Since its beginning, designers have foreseen the need for several classes of service. At the time, the ToS (Type of Service) field of the IP packet was supposed to serve the purpose of packet service identifier. The intentions were never carried to full implementation. Today, ISP (Internet Service Providers) are desperately trying to establish ways to offer guaranteed services to their customers. This effort led to multiple ideas and mechanisms that are still being discussed in standard boardroom meetings. Four principles are presently widely accepted as foundation rules for future QoS provisioning [53]. Their implementation is underway in several standards that were, or will be issued. Each is briefly presented below. In upcoming sections, we'll see how they are put to work in current network services.

***Packet marking and policing*:** This is directly taken from the original approach envisioned by IP designers and allows packets to be distinguished among different classes of traffic. That way, proper processing can be made in network gateways and core systems. In gateways, packets can be modeled to shape the network transmission scheme. In the core, networking equipment can use a packet's tag for easier handling and more precise QoS delivery. But packet marking itself will not guarantee service provisioning. Handling rules, known as *policies*, must also be set in order for routers to distinguish packets and treat them differently. The marking itself is relatively feasible through software upgrades. On the other hand, policy establishment must be agreed amongst all network providers and equipment vendors.

***Traffic isolation*:** Traffic isolation is mainly useful to guard streams against each other's behavior. In some cases, applications' bit rate may not respect the pre-established bandwidth consummation contract and therefore cause starvation to other flows going through the same node. Isolated flows are constrained to a specific bit rate than can't be trespassed. Rate bounding is achieved with scheduling algorithms at network access points and routers. When contract violation occurs, packets can be delayed or dropped, depending on policy rules.

Maximum resource utilization: The strictness of traffic isolation can also lead to resource waste. Once allocated a link, applications can only use granted bandwidth and therefore cannot work with a bandwidth that is allocated to other applications, but not currently used. It is desirable to optimize bandwidth utilization, allowing one flow to use another flow's unused bandwidth at any given moment.

Call admission: Call admission is especially hard to achieve in IP since the base architecture is built on a connectionless, packet switching scheme. Unlike ATM, whose adaptation layers can be configured to model a virtual circuit, IP's architecture can't support end-to-end virtual channels. The basic idea behind call admission is to send a notification message carrying QoS information before exchanging binary content. Once granted, applications are guaranteed that network resources are sufficient enough to accommodate their needs. If resources are insufficient, the call is blocked and returned to the sender with an error mention. Call admission is very difficult to implement because it requires severe changes in networking equipment.

With the above presentation in mind, we now look at practical methods used to materialize these concepts – both in core network equipment and end-systems.

3.2 – Scheduling and Policing Mechanisms:

Scheduling and policing are very important for QoS specification and implementation. The former is a guideline for packet handling at entry points while the latter explicitly controls the way packets are exchanged in the network. One might think that both disciplines are only relevant in router design and network management. However, they are also useful in controlling the output of an application and in performing call admission control at network ingress – two activities directly related with the implementation of the aforementioned QoS principles.

Here, we present the algorithms that are used by Cisco Systems [19,70] in their network equipment. Most of the algorithmic complexity is kept silent since our goal is not to compare them, but to get a general understanding.

3.2.1 – Scheduling:

The most simple and known scheduling model is the classic *FIFO* queue, where arriving packets are stored in a waiting area when transmission devices are busy with other packets. When the waiting area becomes full, packets are dropped in accordance with the packet discarding policy [53]. *FIFO* selects packets for transmission in the same order that they arrive at the input link queue. Although very simple and easy to implement, FIFO does not offer much in terms of QoS. It doesn't support any priority requirements and no guarantees can be made on packet delivery when congestion is high.

Because FIFO was simple, communication companies were not ready to let it go easily. To improve efficiency and provide QoS support, they simply added multiple waiting areas: The *priority queueing*. Each area is targeted to a service requirement. Packet storing is based on the priority class marked in the packet header. When choosing a packet for transmission, the queueing algorithms will transmit packets from the highest priority class that is not empty. The choice among packets in the same class is done in a FIFO manner. The problem with priority queueing is that if a stream associated to a high priority takes all the processing capabilities of the equipment, packets in other waiting areas will suffer from service degradation.

The *Round Robin queueing* architecture addresses the above deficiency by forcing a rotation among waiting queues. Both models are the same, except for the selection of the packet to be transmitted.

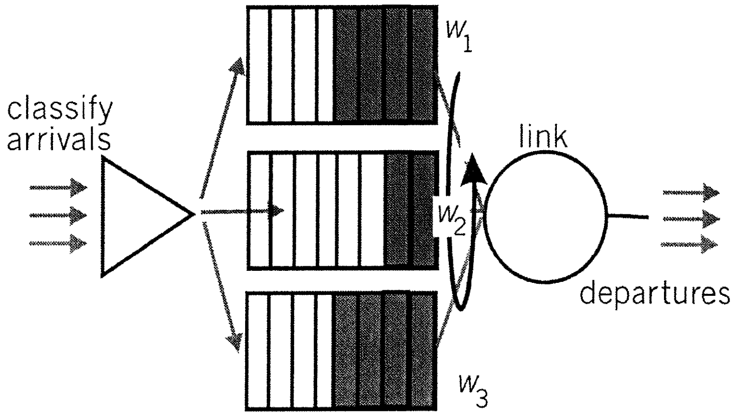


Figure 3.1: Weighted Fair Queueing.

From all algorithms, the *Weighted Fair queueing* (WFQ) [19,87] seems to be the most adequate. It provides fair bandwidth allocation to all network traffic by mixing several properties of the above algorithms. Figure 3.1 shows the architecture. First, packets are classified in queues in accordance with their priority tag. As with the *round robin queueing*, each class is served with different amount of attention. WFQ differs in that each class may receive a different amount of service in any interval of time. More precisely, each class W_i is assigned a weight that has the proportion of $W_i/(\Sigma W_j)$, where the denominator is the summation of all weights associated to each class. The service offered to each queue is directly related to its weight. In the worst case, even if all waiting areas have queued packets, the service will always be honored. This is of prime importance since it allows one to exactly predict the worst time that packets will wait at a node. WFQ plays a central role in QoS architectures and is widely implemented in today's router product.

3.2.2 – Policing:

Policing is one of the cornerstones of any QoS architecture. It is a mechanism for regulating the bit rate that an application is allowed to inject in the network. Many aspects of a data flow can be policed. Three of the most common are the *Average rate*, the *Peak rate* and the *Burst size*. The latter, which is probably the only one that needs clarification, refers to the maximum number of packets that can be sent into the network over an extremely short period of time.

The simplest policing algorithm that encompasses the above mentioned aspects is the *Leaky Bucket* [11,83]. It consists of a bucket that can hold a defined number of b tokens (see figure 3.2). Tokens are always added at a constant rate, let's say r. When the bucket is full, no tokens are added. Before sending a packet, the application must remove one token from the token jar. If it's empty, the application must wait for a new token to be generated. Thus, the token generation rate serves to limit the number of packets that will be sent.

Combining the *Leaky Bucket* and the *Weighted Fair Queueing* gives a provable maximum delay that packets will experience in a queue. At first sight, aggregating both methods may seem troublesome, but no simple solution exists to predict packet delays, especially in IP networks. The last statement is sustained by a number of articles addressing the subject of statistical analysis for approximating delays and jitters [26,52,70]. A very good discussion on the suitability of various approaches for multimedia applications can also be found in [35].

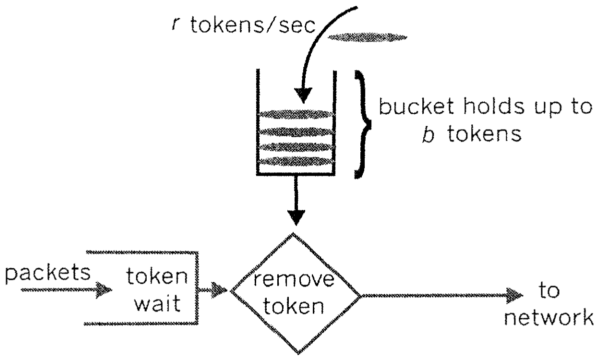


Figure 3.2: The leaky bucket.

It is important to outline that the selection of the above algorithms for QoS implementation is closely tied with the content to be carried. Today's routers and interfaces can be configured to model different types of queueing. When working in a proprietary network, users can easily select and tune their equipment for maximum results. The task of offering the same services when crossing network boundaries is far more difficult. In fact, no assurance or guarantees can be made on the choice of scheduling and policing algorithms that are effective in foreign installations. This is especially obvious in the Internet, since it is an interconnection of networks. Therefore, complex mechanisms are necessary to announce and map the needed services across networks. Clean interoperability can only be achieved through standard protocols and QoS architectures. Due to the number and variability of requirements to address, defining such standards causes serious difficulties.

3.3 – QoS Management Activities:

QoS management ensures that user requirements will be satisfied by their ISP (Internet Service Provider). They can be defined as a set of activities that permit the support, by the Internet service provider, of a desired QoS. When the ISP commits to a requested service, the system must monitor the delivered QoS and take meaningful actions when the established conventions cannot be supported. Depending on the MM application capacities, three actions can be performed: the connection is shut down, the bit rate is adapted to network capacities, or a contract re-negotiation is started.

Service contracts are based on three generic classes of service that encompass all the desired requirements, regardless of their specific characteristics. The *deterministic* *class* [31] offers hard real-time, mathematically provable bounds on delays in the end-to-end path. The *predictive class* [20] is better suited for soft real-time requirements, where time constraints are important, but at a lower level then the previous class. Perdictive services provide guarantees that will fail when the network state becomes too restrictive. Finally, the *best effort class* doesn't make any guarantees about the service to be offered. This means that no matter how important you are and how much money you have, your packets will have to wait in the queues of network routers – typically, this is the case with IP.

Based on the contract that a client has with its ISP, the management functions (activities) to perform are: QoS mapping, admission control, resource reservation, QoS monitoring, QoS adaptation, QoS accounting and QoS policing [39]. Each is briefly depicted below.

3.3.1 – QoS mapping:

The primary role of QoS mapping is to translate service requirements issued by users into meaningful, quantifiable network parameters for ISPs. This step is important since application users understand QoS parameters in qualitative ways that are sometimes far from network metric parameters. For example, the network provider cannot manage the frame rate of an application, but is well aware of the throughput parameter in bits per second. In his work, Hafid [39] defines three types of mapping: *QoS layer mapping*, which formalizes qualitative parameters as they descend the protocol stack, *QoS resource mapping* to derive the amount of resources required to support the requested QoS and *QoS system mapping* that tunes system components (both hardware and software) like synthesizers and image processing device.

Most likely, the result of a mapping process will not accurately reflect the genuine requirements since the number of network parameters are far smaller than the client's qualitative choices.

3.3.2 – Admission Control:

Once formalized, users' QoS parameters are submitted to the admission control procedure that verifies if the requested services can be accepted given the current system load and resource availability. The decision to accept an incoming request is based on the scheduling mechanisms, the characteristics of the data traffic to accommodate and the committed resources. An example of such a process is the classic algorithm found in [57] where a set of m periodic task with processing time p_i and periods of t_i, for $1 \leq i \leq m$, is acceptable if $\sum^{m}_{i=1} (p_i \div t_i) \leq 1$, with $d_i = t_i + p_i$ where d_i indicates the deadline for task i.

3.3.3 – Resource Reservation:

Resource reservation is the step where user requested resources are fetched from network and system equipment. At this point, the admission control process has agreed to accept the newly requested call and thus ensures that QoS can be maintained throughout the end-to-end path. The reservations are made with a signaling call that carries QoS specifications. Reservations can be made in *hard state* or *soft state*. Hard state means that once accepted, a call can only be torn down by a release message whereas in soft state, hosts must periodically refresh the reservation by sending other reservation messages [10,45,52,96].

3.3.4 – QoS Monitoring:

Monitoring plays a very important part in the support of QoS since it informs the system about the current state of the network. It can therefore ask the system to take meaningful actions when quality parameters are shattered. QoS monitoring involves network measurement procedures accounting for parameters such as the number of packet loss, delay and throughput. Such measures must be made available through a network layers protocol such as ATM's management cells or ST-II's feedback mechanism, a functionality that is not provided by IP.

3.3.5 – QoS Adaptation:

When the monitoring activity detects a QoS violation, manipulation on flow characteristics must be made to maintain the agreed contract. QoS adaptation must be capable of handling graceful degradation, reacting adaptively to changes in the environment. The goal is to keep providing the service even if it implies a decrease in quality. Three different techniques are mainly used: interpolation of the missing data caused by packet dropping, dynamic compression techniques that manipulate quantization ratios to reduce the bit rate [34,56,94], and scalable coding [69,87] that decodes an incoming stream based on the different quality levels of the latter and the processing capabilities of the receiving peer.

3.3.6 – QoS Accounting:

QoS accounting is concerned with charging users for whatever resources they're using. It is a key activity since without cost constraints, users would ask for the best QoS available, and thus increase the blocking probability of the system since resources are finite. Accounting is only feasible if communication amongst parties resides in the same network. Otherwise, it is impossible to account for all the billing possibilities caused by network interworking. Up to now and to the best of our knowledge, no proposal has been made to compute and charge for distributed resource usage.

3.3.7 – QoS Policing:

Policing is most likely to be made at the ingress of a network, where user traffic is entering the cloud of nodes. It ensures that an application's bit rate does not go beyond the agreed contract, maliciously or not. This activity prevents sources from affecting each other and thus determinant for ensuring QoS. When an application misbehaves, actions such as packet dropping, user notification, and cost increases can be taken to address the situation. The choice about which action to perform is entirely left to the service provider.

3.4 – Summary:

In this section, we acknowledged the fact that special care must be taken when trying to implement multimedia applications in packet switched networks that are not offering service classes for packet differentiation.

We first saw four basic principles widely recognized as a foundation for providing service classes in IP. *Packet marking*, *traffic isolation, maximum resource utilization* and *call admission control* are the first steps toward QoS implementation. They can be made in core network equipment (traffic isolation, maximum resource utilization) or end-system application (packet marking, call admission control), depending on the expected behavior.

The second part showed some of the enabling algorithms to concretize the above mentioned principles. The key concept to remember is that aggregating the *Weighted Fair Queuing* scheduling algorithm with the *Leaky Bucket* policing method ensures a firm, mathematically provable bound on delays that packets will experience in network switching equipment. Such information is then used by admission control procedures to accept or deny an incoming flow.

We finally had a look at management activities, namely QoS management, that manipulate the characteristics of a stream to model the state of the network. QoS management is responsible for adapting to service variations by gracefully decreasing the perceived quality of a connection. It should also deal with service accounting and billing.

In the next section, we concentrate on the end-points of the end-to-end concept. More precisely, a special type of application that supports real-time constraints to enable the processing of audio and video over networked links is presented. Such application is advantageous in activities like distant learning, entertainment and distributed cooperative work.

4 – Existing Architectures for MM Applications:

4.1 – Existing Architectures for Multimedia Applications:

A Quality of service architecture (QoS architecture) is the gathering of several blocks providing support for MM applications. The goal of a QoS architecture is to define a set of configurable class of services formalizing QoS parameters in the end-system and network, while providing QoS control management and mechanisms [4,26].

The present sub-section provides an overview of actual QoS architectures meeting real-time requirements. Most provide support for the entire distributed system, therefore encompassing both end-systems and network aspects of the QoS. Although proving that QoS in packet switched networks is possible, they sometimes make restrictive assumptions and important tradeoffs that jeopardize their suitability for generic purposes.

4.1.1 – IBM's Heidelberg QoS Platform:

A comprehensive QoS model that provides guarantees in both end-systems and networks had been developed at IBM's European Networking Center in Heidelberg [92].

As depicted in figure 4.1, the communication architecture relies on a continuous media transport system (HieTS/TP) [4,40] that provides QoS mapping and scaling. The network layer of the stack is based on ST-II, a protocol that supports guaranteed and statistical levels of service. The network layer also supports QoS-based routing and filtering through a QoS finder algorithm.

The administration of end-system resources is handled by the resource administration module [40]. The latter is mainly concerned with QoS negotiation, calculation, admission control, network adaptation and resource scheduling. The model was design to serve QoS demands coming from users working over heterogeneous networks in multicast groups. The adaptivity to network fluctuation is supported by filtering and scaling techniques [25] – filtering alters the stream at network ingress whereas scaling matches the source with receivers' QoS capabilities in switches and routers.

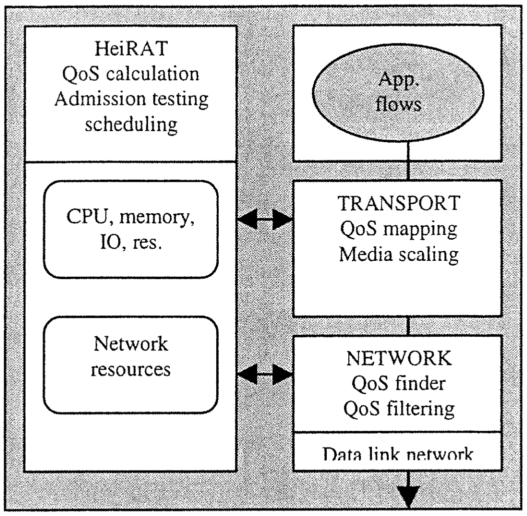


Figure 4.1 : The Heidelberg QoS Model.

4.1.2 – The Tenet Approach:

The Tenet group at University of California at Berkeley [84] designed, implemented and tested the first suite of protocols supporting real-time channels. In particular, these guarantee network performance in terms of throughput, delay, jitters, and reliability with mathematicall provable performance measures [5]. The protocol suite can run on any packet switched network and interoperate with the Internet protocol suite.

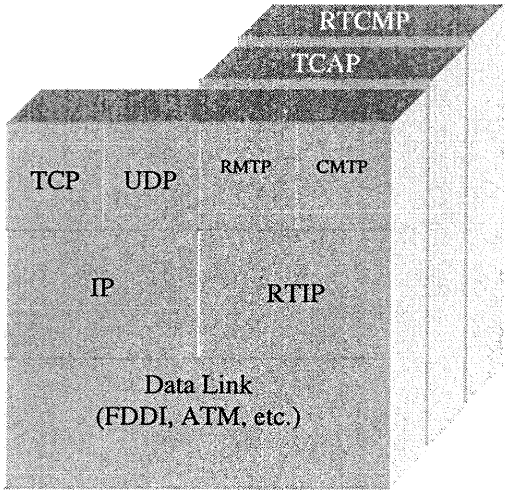


Figure 4.2 : The Tenet Real-Time Protocol Suite [65].

In essence, the Tenet architecture is simply a stack of protocols (figure 4.1) that guarantees performance at all levels. If a layer is incapable of supporting some performance bound, all the above layers will not guarantee that bound either. In all, five protocols provide the desired functionality: the network layer real-time Internetwork Protocol (RTIP); two transport layer protocols – the Real-Time Message Transport Protocol (RMTP) and the Continuous Media Transport Protocol (CMTP) [95]; the Real-Time Channel Administration Protocol (RCAP) for control functions and finally; the Real-Time Control Message Protocol (RTCMP) to detect and recover data transfer failures [32]. Figure 4.2 shows the placement of the protocols in the Tenet stack and their equivalent in terms of Internet protocols.

4.1.3 – Lancaster's QoS-A:

The Quality of Service Architecture (QoS-A) developed at Lancaster University integrates QoS that spans the entire end-to-end path and supports QoS performances for a wide range of applications [17]. Although retaining the best effort model of IP as a special case, QoS-A provides a new class of services (hard and soft end-to-end performance guarantees) through ATM connections. These services were designed to work with highly dynamic environments and thus provide facilities such as performance monitoring, notification of QoS degradation and QoS re-negotiation. The architecture incorporates the following notions: *flows*, which are related to the production, transmission and consuming of single media stream with related QoS; *service contracts*, which are binding agreements of service levels between users and providers; and *flow*

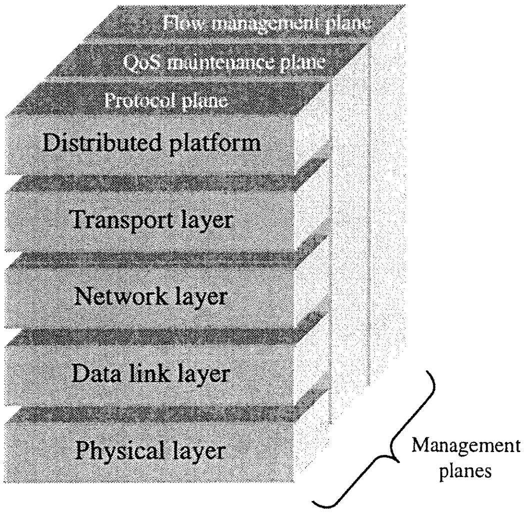


Figure 4.3: Quality of Service Architecture.

management, that provides the network monitoring and maintenance of the contracted QoS level [8,15,16].

QoS-A is built upon other research projects taking place at Lancaster. More precisely, the ATM infrastructure and network interfaces were taken from experimentations of a new class of AAL providing extra services uncommon to actual standard specifications. Furthermore, resource management activities are based on the Chorus micro-kernel that provides scheduling and communication services for continuous media applications [23].

4.2 – Comparison and Discussion:

The QoS architectures presented in section 4.1 are just a sampling of all the work done in the quest for QoS support [4]. All the examples sustain MM requirements through service provisioning made by a client, which is fundamental in capturing application level QoS requirements. With careful examination of individual architecture structures, it is relatively feasible to point out common and diverging aspects.

In most of the work encountered in the literature (and also true with the above cases), the notion of QoS specification is made on a per-flow basis. This means that flows are treated in accordance with their individual characteristics, without concern about what might happen with other flows. In the same way, QoS violations occur on a per-flow basis, regardless of how those violations are treated by the network and/or end-point applications.

To have QoS specification and violation, two crucial assumptions must hold: First, QoS specification must be communicated to stream handling devices through some mechanisms. This is usually made with a *set-up call*, which is basically a message that contains flow requirements used to calculate resource allocation throughout a data path. Second, QoS violation can only be detected if network services can be monitored. The architectures presented earlier performed network monitoring by using either ATM's management cells or a feedback-enable network protocol such as ST-II.

Finally, all architectures somehow dealt with the concept of service agreement specifying the QoS boundaries (the minimum and maximum level of service) received by flows during transmission. Service agreements usually imply that established connections are permanent and that network protocols are "connection oriented".

On the other hand, obvious differences are observed in the type of low level switching scheme. Comet, Tenet and QoS-A rely on ATM's signalling and hard state characteristics to guarantee QoS specifications whereas IBM's Heidelberg QoS platform works with ST-II. The advantages offered by ATM's AAL5 application layer makes it particularly attractive to sustain time-sensitive communications. Indeed, connections are made "on-the-fly" and QoS specifications are easily modelled into ATM's signalling call. ST-II is also a suitable contestant since it provides network feedback with IP-like characteristics. Moreover, ST-II can easily work with conventional IP switches since their header and packet formats are almost alike. Since ST-II is just a network layer protocol, UDP, TCP and other IP-specific protocols can be transported seamlessly.

Some differences between architectures are also noticeable when looking at the signalling protocol used to issue the call set-up. When working with ATM, the solution is trivial since each AAL has its own standard call. In the case of ST-II or other switching schemes, each architecture suggests a different call mechanism that perfectly fits applications' needs.

The options and design choices made in the realization of the four projects are good, valid and logical since they help reach the specified goal of sustaining application level QoS. But these choices also cause tradeoffs that make the architectures either not adapted for today's networks or not feasible in terms of cost and technology availability. Moreover, the number of differences and technologies involved make interoperability almost impossible to achieve.

It is well accepted that ATM networks offer advantages in terms of throughput and built-in QoS that go far beyond all other available network technology. But ATM is very costly and therefore not accessible to the Internet community in general. Also, POTS lines, which are still the main connection link between clients and their ISP, are definitively not capable of sustaining ATM's lowest bit rate (1.5Mbits). As for ST-II, lack of use and support by the industry made it disappear in a shadowy corner of the ITU were it sits under several other experimental protocols waiting to be reviewed for standardisation. This situation leaves us with no real suitable network protocol that can be easily used on a large scale.

Other problems are caused by the specification of the required QoS parameters, which differ from application to application. Divergences in their definition lead us to call request incompatibilities that cannot be encompassed. Even if a consensus about the latter is ever achieved, we'd still have to come up with an agreement on media presentation and cope with interoperability problems in terms of network infrastructure and rendering platforms.

The above-mentioned challenges are simply caused by a lack of standardisation. Standardisation is the only way that enables application interoperability over different networks. Indeed, standard guidelines help application designers to build components that can communicate with other systems coming from different vendors. The ITU envisioned this solution and issued the H.323 standard for MM support in IP networks. Since its approval, legacy application projects as presented above (section 4.1.1 to 4.1.3) were abandoned in favour of H.323 compliant systems.

H.323 is a standard for the presentation of MM content in packet switched networks. It can deal with several network layer protocols and media-coding schemes through a set of conventions and protocols (more on H.323 in section 6). Unfortunately, capability usually implies complexity, which is exactly the case with H.323. Therefore, very few successful implementations have been built outside the industrial community.

4.3 – Summary:

This section gave an insight about distributed multimedia applications and their characteristics.

We first started by classifying them into three broad classes that generally encompasses the available implementations. Applications belonging to the third class, cooperative work tools, are particularly difficult to implement due to the enormous amount of specifications and interacting technologies. Nevertheless, successful implementations have proven the concept feasible and brought interesting results to their users.

Secondly, we looked at MM application requirements and outlined the fact that they significantly differ from conventional designs. High throughput, restrictive time constraints, service commitment and group communication are all aspects to address when designing streaming tools.

The third sub-section draws a close relation between MM application and their supporting architecture, on which relies the burden of dealing and coping with application layer requirements. To guarantee end-to-end QoS, architectures must provision both network and end-system resources. To help grasp the concept, four models developed by leading universities were presented, along with a resume of their features and capabilities.

The next chapter describes the RTP and RTCP protocols. As outlined before, the IP protocol doesn't provide feedback information. To address the deficiency, the above two application layer protocols were introduced. Along with feedback-added capabilities, they were specifically designed to carry time sensitive data. Also presented is the RTSP protocol – a text based protocol commonly used to control and alter the state of media presentations.

5 – Protocols for MM Applications:

We continue our quest for QoS and MM application implementation in IP networks by presenting a set of protocols enabling standard distribution and manipulation of media streams. Since the primary purpose of streaming media is to reach as many people as possible, interoperability issues must be addressed. Interoperability essentially relates to a common representation of some information among all participants, regardless of the tool one uses to manipulate the information.

Hereafter are presented three protocols promoting interoperability. RTP (Real-time Transport Protocol) provides a common way to encapsulate media chunks in data packets to be sent on the network. Its companion control protocol, RTCP (Real-time Transport Control Protocol), allows monitoring of the data delivery in a manner scalable to large multicast groups. It also provides minimal control and identification functionality. Thirdly, the Real-time Streaming Protocol (RTSP) is briefly approached. The latter serves to manipulate media streams with video-like commands, e.g. play/pause.

5.1 – RTP/RTCP Explained:

5.1.1 – RTP Basics:

Since all media streams have a close relation with time, it is convenient to have a packet format that represents time information in a standard way. Other important aspects resulting from media packetization (the segmentation of a media bit stream in a series of disjoint packets) could also benefit from a common representation. The RTP protocol, defined in RFC 1889 [76] and RFC 1890 [77], offers a standard packet structure that includes sequence numbers, timestamps, payload identification, and synchronization information. Through these mechanisms RTP provides end-to-end transport for real-time data over datagram network, a feature that made the protocol an important part of the IETF Internet telephony architecture [80].

Although not making any discrimination about the transport protocol to be used, RTP typically runs on top of UDP (User Datagram Protocol) [18]. Audio and video chunks of media, generated by the sending side, are packaged in RTP packets which are in turn encapsulated in UDP segments. Since RTP provides services to multimedia applications (timestamps, sequence numbers, etc.), it can be viewed as a sublayer of the transport layer (figure 5.1a). From a developer's point of view though, RTP is not included in the standard IP protocol stack and therefore not part of the socket API (Application Programming Interface).

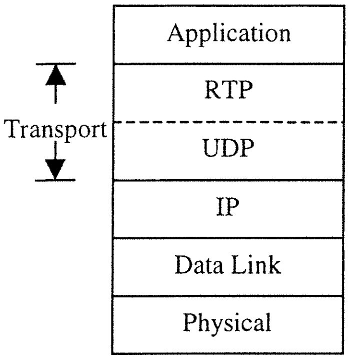


Figure 5.1a: RTP viewed as a sub-layer of layer 3.

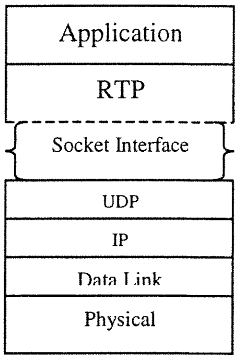


Figure 5.1b: RTP as part of the application layer.

RTP packets are integrated at the application layer where developers must merge the protocol with a media codec. The resulting packets are sent into a UDP socket interface. Similarly, at the recipient side, RTP segments enter the application through the UDP interface and thus, coded instructions must be written to extract media information from the packet (figure 5.1b). Taken from [58], the following list summarizes some of its features.

1. RTP itself does not provide any mechanism to ensure timely delivery. It needs support from lower protocols having control over end-to-end resources such as RSVP for signalling and UDP for data transport.
2. RTP doesn't assume anything about the underlying network, except that it provides framing. RTP typically runs on top of UDP to make use of its multiplexing and checksum services.

3. RTP doesn't offer any form of reliability or flow/congestion control. It provides timestamps and sequence numbers as hooks for adding reliability and flow control, but implementation is entirely left to designers. [12,28].
4. RTP is a protocol framework deliberately incomplete. It is open to new payload formats and multimedia software. Each supported media comes with an extra standard document describing the correct implementation, e.g. [44].
5. RTP only carries real-time content and is not responsible for high-level tasks like stream assembly and synchronization. These must be done at the application level.

5.1.2 – RTP Packet Format:

As mentioned previously, the RTP format provides a standard representation of media-dependant information usually needed for coding and decoding audio/video streams. As defined in [58], a packet header has the following format:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|V=2|P|X|  CC   |M|     PT      |       sequence number         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         RTP timestamp                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           synchronization source (SSRC) identifier            |
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
|            contributing source (CSRC) identifiers             |
|                             ....                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 5.2 : RTP Packet Header.

The first twelve octets are present in every RTP packet, while the list of CSRC (Contributing Senders) identifiers are present only when flows are mixed together. The fields have the following signification:

Version (V): 2 bits. The version of RTP, which is actually in its second revision.

Padding (P): 1bit. If set, the end of the packet contains one or more additional padding octet(s) that are not part of the payload. The last octet of the padding contains the number of octet to ignore.

Extension (X): 1 bit. If set, the fixed header is followed by exactly one header extension. Extensions are needed when adding proprietary features.

CSRC count (CC): 4 bits. The number of CSRC identifiers following the fixed header. This number is more than one if the payload contains data from several senders.

Marker (M): 1 bit. Defined by a profile, the marker is intended to allow significant events such as frame boundaries to be marked in media streams.

Payload type (PT): 7 bits. Identifies the format of the RTP payload (H.261, Wav, MPEG) and determine its interpretation by the application.

Sequence Number: 16 bits. Incremented by one for each data packet sent. It may also be used by receivers to detect packet loss and to restore packet sequence.

Timestamp: 32 bits. The sampling instant of the first octet in the RTP data packet. The time representation is based on the carried data format and may *not* reflect wallclock[1] time.

SSRC: 32 bits. The Source Synchronization identifier gives a random and unique number to all participants in the same session. It servers to uniquely detect the sender of a stream.

CSRC list: 32 bits each. The contributing source(s) for the payload contained in the packet. The number of identifier(s) is given by the CC field and limited to 15.

Aside from media support, RTP was found to be useful for many kinds of application imposing hard time constraints such as distributed simulation, industrial assembly lines and critical communication systems. More details about RTP and RTCP (the latter coming up next) can be found on their creator's web page and other related sources [22,65,79].

[1] The time representation known and used in every day purposes, e.g. HH:MM:SS.*MM*

5.1.3 – RTCP Control Protocol Basics:

RTCP constitutes the second part of RFC 1889 [76] and is designed to work closely with RTP. It conveys control information based on the periodic transmission of packets to all participants in a session using the same distribution mechanism. It performs four functions:

1. It provides feedback on the quality of the data distribution, for both senders and receivers.
2. It carries sufficient information to identify each participant in a session with other means than the SSRC.
3. It enables the scaling of sessions by making the number of participants in the exchange available.
4. The fourth and optional function is to bring minimal session control information.

Several kinds of control reports are specified to carry a variety of management information. In all, five types are defined: 1) SR (sender report), for the transmission and reception of statistics from participants that are active senders; 2) RR (receiver report), for receiving statistics from passive participants; 3) SDES (source description item) that contains the information about a given participant; 4) BYE indicates the termination of participation and finally; 5) the APP, which specifies application specific functions. Each RTCP packet begins with a fixed header similar to that of RTP, followed by specific control related information.

5.1.4 – RTCP Packet Format and Monitoring Capabilities:

Among the above-described reports, the sender (SR) and receiver (RR) types are particularly important since they provide the reception quality feedback. Both are alike except for an extra 20 byte included in the SR report. Figure 5.3 depicts such a report and shows how the information can be used to compute monitoring statistics.

Only the most important fields are described to lighten the presentation. Details on the missing contents can be found in section 5.1.2 or RFC 1889 [76].

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|V=2|P|    RC   |   PT=SR=200   |             length            | header
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         SSRC of sender                        |
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
|              NTP timestamp, most significant word             | sender
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ info
|             NTP timestamp, least significant word             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         RTP timestamp                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     sender's packet count                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      sender's octet count                     |
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
|                 SSRC_1 (SSRC of first source)                 | report
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ block
| fraction lost |       cumulative number of packets lost       |   1
-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           extended highest sequence number received           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      interarrival jitter                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         last SR (LSR)                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   delay since last SR (DLSR)                  |
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
|                 SSRC_2 (SSRC of second source)                | report
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ block
:                               ...                             :   2
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
|                  profile-specific extensions                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 5.3 : Sender report RTCP packet.

Many fields presented in figure 5.3 are common to those of RTP's header. A few are relatively easy to figure (packet count, octet count and lost measures) thanks to their explicit name. Here is an insight about the remainings.

NTP timestamps: 32 bits each. Time representation in RTP/RTCP protocols follow the international *Network Time Protocol* (NTP) [63] standard. According to the latter, time is presented with a 64 bit structure. The most significant word presents time to a precision of a second, while the least significant word is for milliseconds.

Last SR timestamp (LSR): 32bits. The middle 32 bits out of 64 in the NTP timestamp [63] received as part of the most recent RTCP sender report (SR) from the souce SSRC_n. It is used, along with the DLSR (see below), in the round trip time computation (RTT).

Delay since last SR (DLSR): 32 bits. The delay, expressed in units of 1/65536 seconds, between receiving the last SR packet from source SSRC_n and sending this reception report block. Taken from [76], figure 5.4 shows the relation between LSR and DLSR and how both measures are used.

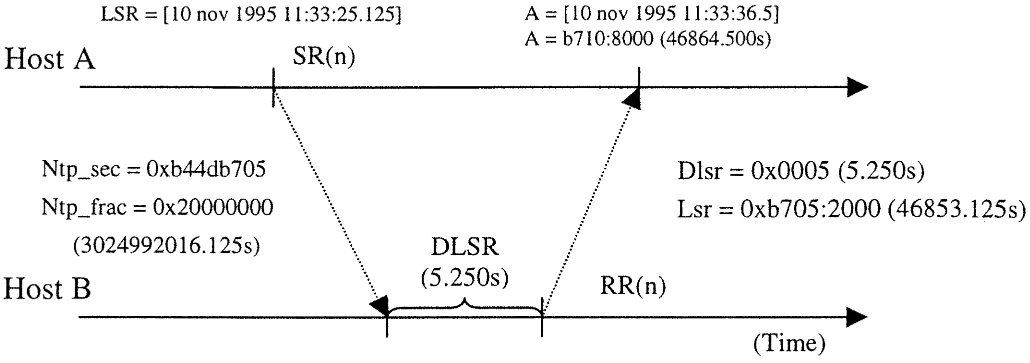


Figure 5.4: LSR and DLSR explained.

LSR and DLSR are very useful to evaluate the throughput yielded by links between communicating peers. In accordance with figure 5.4, the round trip time can be computed as follow:

$$RTT =(A - (DLSR + LSR)) \qquad [1]$$

From equation [1], it is easy to figure that *RTT/2* is the approximative time taken by the RTCP packet to travel from host B to host A. The throughput of the link, for an RTCP packet is thus:

$$Throughput_{RTCP} = RTCP_{packet\,AVG.} / (RTT/2) \qquad [2]$$

where $RTCP_{packet\,AVG}$ is the average size of a receiver report. Combining equation [1] and [2] gives the estimated throughpout for RTP packets:

$$Throughput_{RTP} = (RTP_{packet\,AVG.} \bullet Throughput_{RTCP}) / RTP_{packet\,AVG.} \quad [3]$$

Even if the bandwidth calulation of equation [3] is obviously not precise, it gives enough information about the state of a link which can later be used to adapt an application's bit rate. In most cases, imprecisions introduced by the above computation are not significant since media streams are jitter prone by nature.

Interarrival jitter: 32 bits. An estimate of the statistical variance of RTP packet interarrival time, measured in media-time units. This interarrival value is the mean deviation (smoothed absolute value to minimize fluctuations) of the difference D in packet spacing.

For each two packets i and j arriving, the value D is computed:

$$D\ (i,j) = (R_j - R_i) - (S_j - S_i) \qquad [4]$$

where S_i is the RTP timestamp from packet i, and R_i is the time of arrival (still in timestamp units) of packet i. Equation [4] is performed continuously as each data packet i is received from a sender, using the difference D of that packet and the previous packet $(i - 1)$, yeilding:

$$J = J + (\ |\ D(i - 1,i)\ |\ - J) / 16 \qquad [5]$$

The value of J is sampled each time a reception report is issued. This algorithm is the optimal first-order estimator and the $1/16$ gain gives a good noise reduction ratio while keeping a reasonable convergence rate [13].

Given the information presented above, sending and receiving applications can compute many useful network parameters. The interarrival jitter provides a short-term measure of congestion. Packet loss tracks persistent congestion while jitter measures track transient congestion. Throughput measures can be combined to depict long-terms network performance used by congestion pridiction algothims. Finally, RTCP reports are useful to compute the number of participants in a session. The result is then used to scale the amount of feedback information with the size of a session [73] to avoid network flooding by feedback packets.

5.2 – RTSP:

The Real Time Streaming Protocol is an Internet Engineering Task Force standard for the control of streaming media on the Internet [78]. It essentially provides an extensible framework to manipulate on-demand delivery of real-time data such as audio and video coming from live feeds or digital storage. The protocol is intended to handle multiple data delivery sessions and provide a means for choosing channels like UDP, multicast UDP or TCP.

The idea behind RTSP was to design a protocol that could include sufficient amounts of vocabulary allowing users to control the state of dynamic data flows. The base principle is the same as HTTP. The two fundamentally differ in a couple of aspects though. First, RTSP introduces many new methods and has a different protocol identifier. Since it can control more than one stream at the time, RTSP servers must keep track of individual stream through state machines. Second, RTSP is working "out-of-band" and thus, packets are carried on a different channel than multimedia data. Furthermore, RTSP can be carried on any kind of transport protocol since it doesn't impose time constraints.

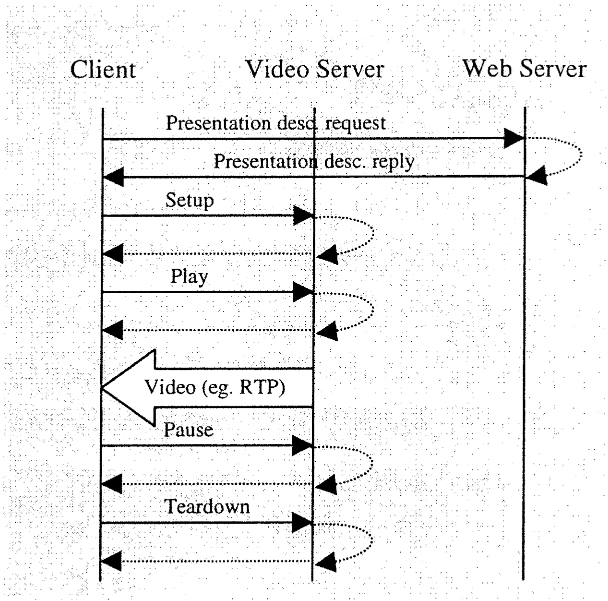


Figure 5.5: Typical RTSP operations [85].

In short, RTSP is only concerned with stream identification and control. Thus, the retrieval of media from a server, the invitation of a server to a conference and the addition of media to an existing presentation (showed in figure 5.5) consitute its main capabilities. Finally, it is extendable, secure, easy to parse, and transport independent. The company RealNetworks, its main promoter, keeps a wealth of information about the protocol on its web site [74].

5.3 – Summary:

The intention of this chapter was to present different protocols defined for the support of real-time requirements. As depicted, implementers usually have to provide support and algorithms since standard documents are only concerned with format specification.

The first two protocols have been designed to bring a common agreement on packet format tailored to the transport of time-sensitive information. As such, RTP provides sequence numbers, timestamps, payload identification, and synchronization information which are believed to have a common use in all stream transport protocol. Its related control protocol, RTCP, provides the network feedback feature that is not supported by IP. With the conveyed reception report, it is possible to have network sampling measures such as the number of packet loss, an approximation of the throughput and an evaluation of jitters introduced by each step of a data path. Nowadays, RTP and RTCP are both accepted protocols widely used for the transport of audio and video in packet switched networks.

Third and last was RTSP, a text-based protocol allowing for the identification and control of media streams. Even if RTSP is currently implemented in major multimedia applications, its future may not be guaranteed. Another efficient and widely used stream control mechanism, the DMS-CC (*Digital Media Storage – Command and Control)*, was released with the MPEG-2 standard as part of the new features included with the compression scheme. With the rapid acceptance of MPEG-2 for A/V compression, DMS-CC gained popularity and hot debates on which should be used are still frequent in the multimedia community.

In the upcoming section, the H.323 suite is presented. It is an architecture consisting of components and standard protocols (some of which were presented above) allowing a clear and open definition of guidelines for designing interoperable MM applications. It is supported by most industry players and believed to represent tomorrows streaming standard for all implementations.

6 – The H.323 Standard for MM Applications:

The last five chapters described different aspects and knowledge about QoS and MM applications. In fact, each topic was a building block of an overall technology that enables media streaming in packet based networks that don't support QoS by any means.

This section combines the material viewed in previous discussions to present the H.323 standard for MM application. H.323 is a specification engineered by ITU's Study Group 16 to describe terminal, equipment, and services for media exchange over packet based networks such as the Internet. It is an aggregate of different interworking technologies addressing call control, media management, bandwidth management and interfaces between LANs and other networks. To this day, H.323 is the most complete, efficient, and interoperable architecture. Since the beginning, it has been accepted and supported by many industry leaders.

The following presents a summary of the H.323 standard. We first start by introducing the model's general features. Next we depict the main protocols for audio and video streaming. The section closes by making a brief look at the components of a complete H.323 system.

6.1 – H.323 Introduced:

As defined in [47], H.323 is a standard that specifies the components, protocols and procedures that provide multimedia communication services – real-time audio, video, and data communications – over packet networks. In addition to media conferencing terminals and other related equipment, the specification describes interoperation of H.323 systems with other audio/video conferencing tools on ISDN (Integrated Service Digital Networks), POTS (Plain Old Telephone Systems), ATM, and other LAN technologies [86]. It provides a wide range of services and can therefore be applied to a large variety of areas such as consumer, business and entertainment applications.

The H.323 recommendation is comprehensive and flexible. It can be applied to voice-only handsets and full multimedia video-conferencing stations. Among its many features, the following are particularly interesting [24]:

Codec Standard: H.323 establishes standards for compression and decompression of audio and video data streams, ensuring that equipment from different vendors will have some area of common support.

Interoperability: Besides ensuring that receivers can decompress the information, it describes methods for allowing clients to communicate their capabilities to other participants.

Network Independence: H.323 is designed to run on top of common network architectures. As network technology evolves, and as bandwidth management techniques improve, the actual based solutions should take advantage of those enhanced capabilities.

Platform and Application Independence: The standard is not tied to any hardware or operating system. It can be implemented in any platform such as video-enabled personal computers, dedicated platforms, IP-enabled telephone handsets and other embedded devices.

Inter-Network Conferencing: H.323 uses a *gateway* to link different network technologies together. It uses codec from several videoconferencing standards to minimize transcoding delays and provide optimum performance.

Other key benefits like multipoint support, bandwidth management, multicast support and implementation flexibility are also considered as prime technology advancement.

6.2 – H.323 Architecture Protocols:

The H.323 standard uses an assortment of protocol that ensures standard communication control over four critical aspects: *call control functions*, *audio*, *video* and *data exchange*. Communication in the stack is considered to be a mix of the above. Audio capabilities, Q.931 call setup, RAS (Registration/Admission/Status) control and H.245 signalling are required (figure 6.1). All other features such as video and data conferencing are optional. Finally, the RTP/RTCP protocols form the base support for encapsulation and transmission of media streams.

Inspired from [24,86,93], we describe below the four mentioned aspects in terms of functionality and protocols pertaining to one another. Where possible, the relation between protocols and how they compare to those of section 5 will be outlined.

6.2.1 – Call Control Functions:

The call control functions are the heart of the H.323 terminal. These functions include signalling for call setup; capability exchange; signalling of commands and indications; and messages to open and describe the content of logical channels. All audio, video, and control signals pass through a control layer that formats data streams into messages for output to the network interface. The reverse process takes place for incoming streams.

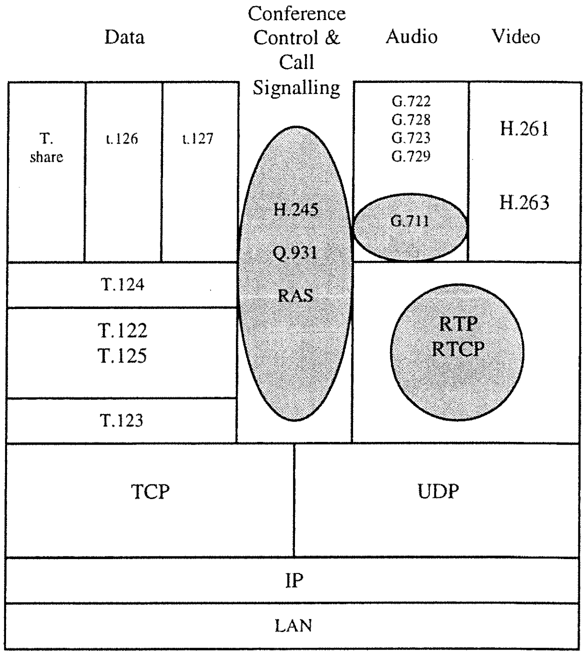


Figure 6.1: H.323 protocols. Mandatory ▨

Overall system control is provided by three separate signalling functions: the H.245 Control Channel, the Q.931 Call Signalling Channel, and the RAS Channel. The H.245 Control Channel is a reliable connection that carries control messages governing operation of the H.323 entity including capability exchanges; opening and closing of logical channels; preference requests; flow control messages; and general commands and indications. Capability exchanges are enabled by H.245 that provides for separate receivers and transmit capabilities. Methods to describe those details to other H.323 terminals are also defined. There is only one H.245 Control Channel per call.

The Call Signalling Channel uses Q.931 to establish a connection between two terminals. The RAS signalling function performs registration, admission, bandwidth changes, status, and disengage procedures between endpoints and Gatekeepers. RAS is not used if a Gatekeeper is not present.

The control capabilities expressed in H.323 are far more advanced than RSVP or even DMS-CC. This is caused by amount equipment managed and the complexity inherent in the exchange of audio, video and data at the application level.

6.2.2 – Audio:

Audio signals contain digitized and compressed speech. The compression algorithms supported by H.323 are all proven ITU standards. Terminals must at least support the G.711 voice standard for speech compression. Other voice support standards are optional.

The different ITU recommendations for digitizing and compressing speech signals reflect variable tradeoffs between speech quality, bit rate, computer power, and signal delay. G.711, originally designed for continuous bit-rate networks, generally transmits voice at 56 or 64 kbps – which is well within the bandwidth limit likely to be found on a LAN.

6.2.3 – Video:

While video capabilities are optional, any video-enabled terminal must support the H.261 codec; support for H.263 is optional. Video information is transmitted at a rate no greater than that selected during the capability exchange. H.261, which provides a measure of compatibility across different ITU recommendations (sub-QCIF, QCIF, CIF, 4CIF, 16 CIF), is used with communication channels that are multiples of $p \times 64$ kbps (p = 1,2,3...30). H.261's encoding algorithm is based on the discrete cosine transform (DCT) and makes use of intra-frame coding. Motion compensation vectoring, which improves image quality, is an option [46,56].

H.263 is a backward-compatible updated to H.261. H.263's picture quality is greatly improved by using a ½ pixel motion estimation technique, predicted frame coding, and a Huffman coding table optimized for low bit rate transmissions. H.263 defines five standardized picture formats, including QCIF. This common support facilitates communications between H.261 and H.263 systems [64].

6.2.4 – Data:

Data conferencing is an optional feature. When supported, it enables collaboration through applications such as shared whiteboards, application sharing, and file transfer.

H.323 supports data conferencing through the T.120 specification (Figure 6.1). As an ITU standard, T.120 addresses point-to-point and multipoint data conferences. It provides interoperability at the application, network, and transport level with the T.12*X* set of protocol.

6.3 – H.323 Architecture Components:

The standard specifies four kinds of components, which, when networked together, provide point-to-point and point-to-multipoint communication services. As shown in figure 6.2, these components are physical entities located in a "H.323 zone". Here, a zone is the collection of all *terminals*, *gateways* and *MCUs* (Multipoint Control Units) managed by a single *gatekeeper*.

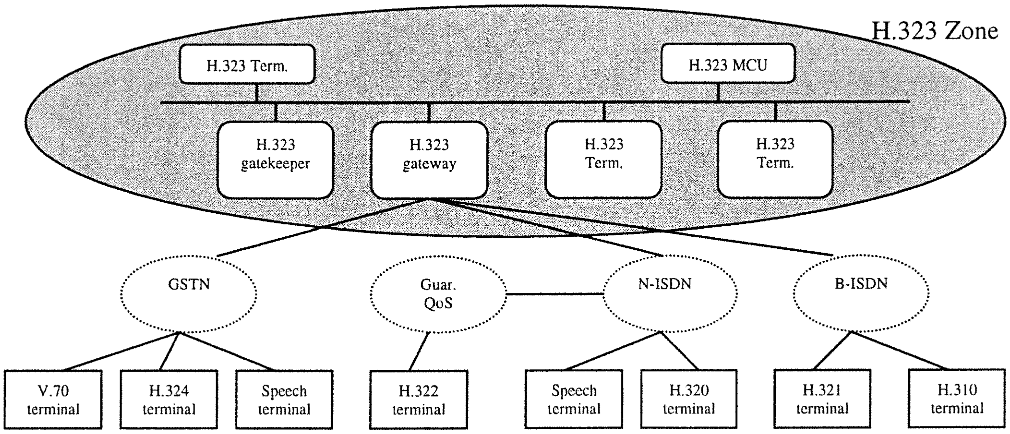


Figure 6.2 : The H.323 Architecutre Components.

Terminals: Used for real-time bi-directional multimedia communication. It can be applied to any system capable of running the H.323 protocol stack (see section 6.1) with the required capabilities. It should also interact with H.324 on SCN and wireless networks, H.320 terminals on N-ISDN, H.321 on B-ISDN and H.322 terminals on guaranteed QoS LANs.

Gateways: A gateway connects two dissimilar networks. It provides connectivity between an H.323 and a non-H.323 network. This connectivity of incompatible technologies is achieved by translating protocols for call setup and release, converting media formats, and transferring information among networks connected by the gateway. The presence of a gateway is not required for communication among parties using the same network technology.

Gatekeepers: The gatekeeper is the most important component of an H.323 enabled network. It acts as the central point for all calls within its zone and provides call control services to registered endpoints. Although not mandatory, it performs important management services such as addressing, authorization and authentication of terminals and gateways; bandwidth management; accounting; billing; and charging. Figure 6.3 lists the required base functionalities. If present in a zone, terminals must make use of the services offered by the gatekeeper. In this case, RAS protocols (refer to 6.2) are used to perform mandatory management activities.

Address Translation	Translation of alias address to transport address using a table that is updated with registration messages.
Admission Control	Authorization of LAN access using ARQ, ARC and ARJ messages. Access may be based on call authorization, bandwidth, or some other criteria.
Bandwidth Control	Support for bandwidth request, confirm and reject messages (BRQ, BCF, BRJ).
Zone Management	The gatekeeper provides the above functions for terminals, MCUs, and gateways that have registered within its zone control.

Figure 6.3: Gatekeeper's mandatory functions.

MCU (Multipoint Control Units): The MCU supports conferences between three or more endpoints. Under H.323, an MCU consists of a required Multipoint Controller (MC) and zero or more Multipoint Processors (MP). MC and MPs are entities enabling *centralized* and *decentralized multipoint* conferences. All terminals participating in the conference establish a connection with the MCU. The latter manages conference resources, negotiates between terminals to determine the A/V codecs to use, and may handle media streams.

The second version of H.323 was approved in 1998 and addresses deficiencies found in version 1. It introduces new functionality within existing protocols, such as Q.931, H.245 and H.225. The most significant advances were made in security, fast call setup, supplementary services and the aggregation of T.120 with H.323 protocols [24]. The standard video codec H.263 was also added to provide enhanced picture quality in videoconferencing application working on high bandwidth networks.

6.4 – Summary:

H.323 compliant systems offer a practical and realistic solution to MM streaming in packet base networks. Moreover, the ability to work with heterogeneous technologies, both in terms of network and platform, make H.323 much more efficient and appropriate than any of the architectures of section 4.

H.323 is a set of architecture components and protocols working together to establish a framework viable for real-time data streaming. It tackles problems pertaining to time constraints and network deficiencies by providing a means for client application to select parameters that are best suited for their environment. It also delivers a standard set of technology requirements (codec, protocol, signalling) that ensures common understanding among all participants.

The idea behind the recommendation is also stronger than the changes introduced in the switching models of section 3. Recall that IntServ [9,10,96,97], DiffServ [85,99] and MPLS [2,14,43,72] all modified the present Internet switching scheme by differentiating traffic in core network equipment. Instead, H.323 tries to live with network gaps by adapting to the environment, which is a more suitable solution for acceptance and implementation.

H.323 is undoubtedly powerful enough to undertake tomorrow's multimedia challenges. Even if the amount of realization requirements and the number of specification (as outlined in this section) cause a serious threat to rapid deployment, many serious players have begun its implementation for various devices, ranging from personal computers to hand held devices.

This section concludes the material to be reviewed for the implementation of an end-system QoS aware protocol stack. Next we present a solution that introduces QoS adaptation in error prone environments.

7 – The Suggested Architecture:

This section presents a QoS architecture designed in a joint effort by three students – Jean-Marc Ng Wing Keng, Yong Guo and Mathieu Poirier. The project is motivated by distant learning and implies the distribution of time sensitive media on IP networks. To solve the constraints added by heavy-streamed media and promote tool interoperability, the ITU's H.323 protocol stack of section 6 constitutes the foundation of our work. The goal is to produce an architecture that is both compliant to H.323 and capable of offering application level QoS specification.

To achieve the above goals, we bring several new modules to the existing stack. The purpose of the added components is threefold. First, we want to introduce an easier QoS representation and specification process, both for clients and servers. This feature should help users choose media parameters more easily. Second, we recognized that media session should adapt to network service fluctuations in agreement with users' preferences. Modifications are made to the original H.323 stack since it's not concerned with such matters. Finally, the novel idea of interacting with several transport layers (ATM, IP, Wireless) is added.

The upcoming material is organised as follows: First, we picture the global architecture in its environment and depict the general behaviour and functionality of each component. The second part discusses about the assumptions and tradeoffs that make the realization of a prototype feasible. To avoid loosing sight of our primary concerns, several abstractions have been applied and priority given to QoS enabling modules.

7.1 – Architecture and Components:

We hereby present the design of our QoS stack by describing the added modules and their interactions with other components of the system. We concentrate on the new functionality and assume that H.323 basic knowledge has been acquired in section 6. But before dwelling on the heart of the subject, let's look at the main features brought together by our work.

- Since it's based on H.323, all the characteristics and advantages of the latter are inherited.
- It can handle several network interfaces seamlessly, which is desirable in exploiting different features offered by available network technologies.
- Allows for an easier QoS representation and specification by presenting parameters that can be understood by common users.
- Automatic translation of the above-mentioned user comprehensible parameters to network specific values and bounds.
- Capable of handling data, audio and video streams in accordance with users' QoS specifications. These are also used to modify connection properties when network services fluctuate.
- Monitoring of each flow separately to guarantee better responses and independence among ongoing connections.

We enhanced the H.323 standard with two main modules: one (the *QoS Manager*) that considers user requirements and network status to take actions that degrade or improve the perceived QoS and another (the *Transport Controller*), to control interactions with the available network interfaces. The dark-greyed blocks in figure 7.1 highlight these modules. The remaining coloured blocks have also been added to support the desired services. Here is the description and purpose of each component.

7.1.1 – The QoS Manager:

The *Quality of Service Manager* (QoS Manager) – designed and implemented by Yong Guo – is the 'intelligent' entity responsible for adapting stream characteristics to network changes based on user requirements. To do so, it refers to a set of predefined instructions clustered in a file called *user profile*. Each user

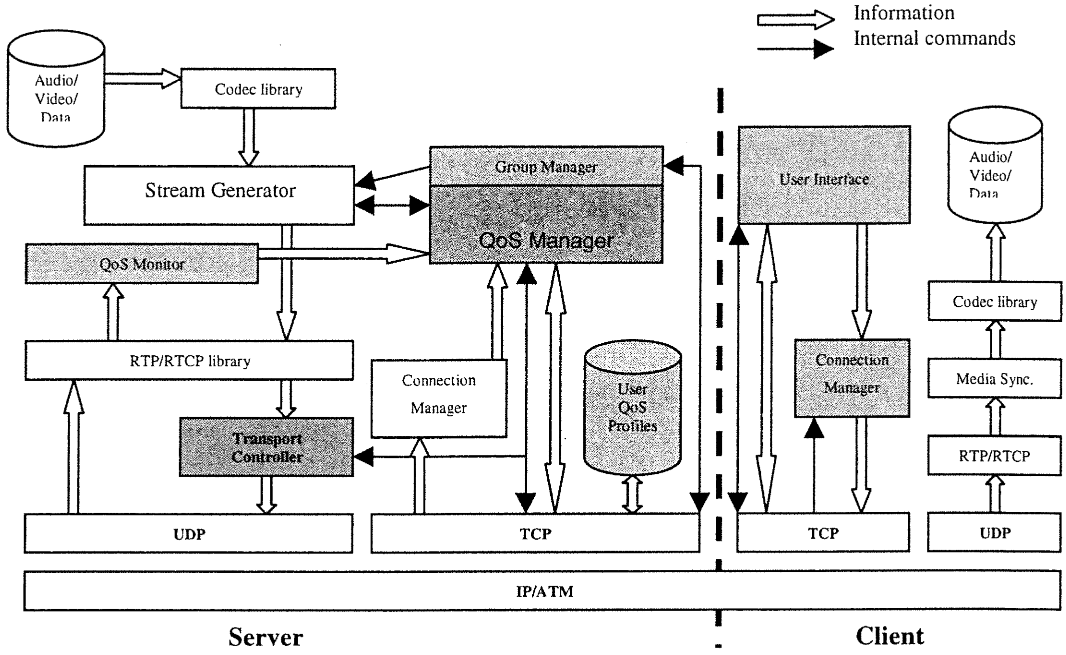


Figure 7.1: An improved H.323 Stack.

involved in media exchanges must have such a *user profile* to guide the *QoS Manager* in its decision. Profiles are independent of one another, which guarantees independence between clients.

A *user profile* forms part of a series of recommendations issued by a client. Those recommendations specify the actions to follow upon network service fluctuations. Also referred as *adaptation strategies*, recommendations can instruct the manager to close a media connection when network throughput goes below a given threshold or when the number of packet loss becomes too important. Less drastic measures can include the reduction of the frame rate or an increase of the compression level to reduce the required bit rate.

The *QoS Manager* takes network state information from the *Stream Generator* who is responsible for computing such data. Information updates are made more than four times per second. Whenever the *QoS Manager* receives an update, it refers to the user profile associated with the connection. If a QoS violation (a level of service that is inadequate for media transmission) occurs, actions are taken based on user specifications to properly adapt the ongoing presentation.

7.1.2 – The Transport Controller:

The transport layer – design and implemented by Jean-Marc Ng Wing Keng – in this architecture is somehow special. Whereas classic transport layers are concerned with a single network, ours can handle multiple network interfaces. This means that the stack can communicate with distant clients over different networks such as IP, ATM, ISDN or others. The type of network itself is not important – most significant are the services offered by a network based on users' needs.

A client can therefore participate in media exchanges over more than one network, following the requirements for each media. If a connection requires high throughput and low packet loss, ATM or ISDN network will be favoured. On the other hand, for bulk data transfer, a less costly network such as IP should be used. Obviously, the cost factor is the only aspect that will prevent everyone from choosing high quality connections. In that context, we assume the system is working in an economically balanced environment where offer and demand levels are established.

The network chosen for media transport is governed by the *QoS Manager* described in the above section. The latter takes its decision after user specification. The manager can also instruct to change the network onto which a media is streamed if the actual network quality becomes insufficient. This kind of network switching is also submitted to a billing process and as such, the Manager should always make sure that it chooses a connection that maximizes the cost/efficiency ratio.

Within the layer and for each network, different queues separate audio, video and data packets. The added level of refinement enables to favour a media over another based on its importance for the client. Typically, audio has the prime consideration, followed by video and then data. Clients can always change this order and the amount of bandwidth given to each media in their user profile.

7.1.3 – The Connection Manager:

The *Connection Manager* – designed and implemented by the author – is mainly concerned with call establishment, bit rate negotiation, media parameters and the selection of rendering tools. More generally, it makes sure that each entity involved in a media exchange agrees on the parameters to be used during the session. This module is not part of the standard H.323 systems and was introduced to bypass the complexity inherent to using H.225 and H.245 connection establishment protocols. To this date, the connection manager is equipped with basic call establishment capabilities. Although not used, provisioning had been made to include rate negotiation, media parameters and rendering tool selection in the request protocol.

Recall that in H.323, the above functionality is achieved by using two protocols, H.245 and H.225. The H.245 is mainly concerned with dialogue between the gatekeeper, clients and servers. It is also mandatory when calling from another network than IP or when using the *Lightweight Directory Access Protocol* (LDAP) for interaction with the *Plain Telephone Switched Network.*

H.225 is specifically charged to negotiate port selection, media codec, and machine capabilities between peers. The might of this protocol is shadowed by the complexity inherent to several tuning and dialogue features included in the specification. To overcome the problem and to introduce application specific mnemonic exchanges, we have designed a home bred protocol. This deviation from the standard specification automatically poses interoperability issues that shall be addressed if conformance tests are performed.

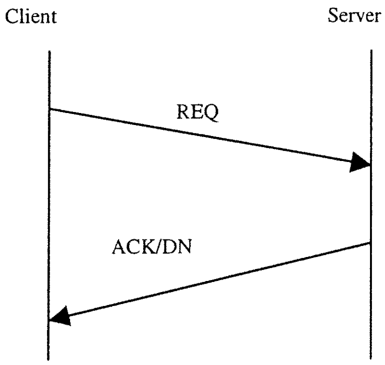


Figure 7.2: Call establishment procedure.

This new protocol is composed of three messages: *REQ*, *ACK* and *DN* (please refer to Figure 7.2). The first is used to request a connection from client to server. *ACK* and *DN* are the answer to that request. The former is a positive reply and includes the necessary parameters that will complete the call establishment. After the reception of such an acknowledgement, media streaming can begin.

DN is a denial response and includes a message explaining the reason for the turndown.

Each media needs a call request. Such design patterns allow every connection to have unique characteristics, which are taken care of by the server on a client basis. Characteristics are specified in a user profile that is included in the *REQ* message along with other connection specificity.

Figure 7.3 depicts the packet format and information carried in each message. The first two fields are common to each mnemonic. The former is a packet identifier indicating which processing should be made upon reception.

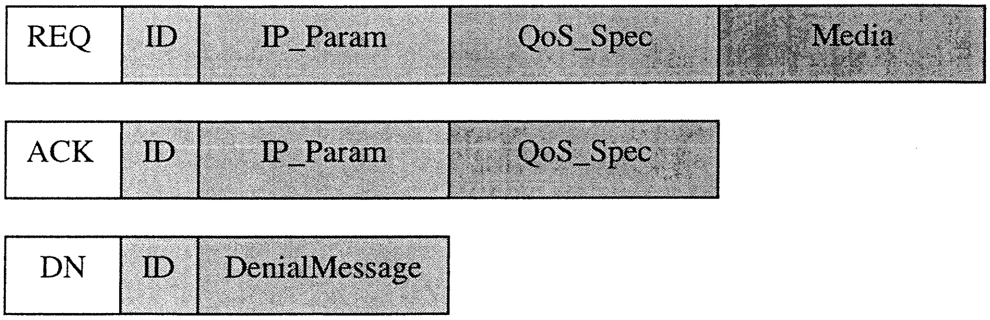


Figure 7.3 : Connection Packets.

The *ID* field enables the marking of call establishment. Each is unique and permits an asynchronous reply from the server. Therefore, the order in which connections are requested and answered is not automatically the same. This can be explained by the connection acceptance process performed by the *QoS Manager*. Depending on the requirements and the type of link requested by a user, connection answer time can differ. *IP_Param* is the gathering of all information needed to establish a media session such as port numbers for RTP/RTCP, IP addresses, session name and so on. User defined preferences are packaged in the *QoS_Spec* which is handled by the *QoS Manager* once received at the server side. Finally, the *Media* field conveys information about the requested media to be played.

7.1.4 – The Stream Generator:

The *Stream Generator* – designed and implemented by the author – is usually called *Streamer*. It was introduced in the architecture (figure 7.1) to simulate the acquisition and packetization of media information. This is motivated by the fact that H.323 compliant codecs are retailed at exaggerated prices going beyond our capabilities. In H.323 compliant systems though, media acquisition, compression and packetization is made at the application level by dedicated multimedia applications enabled with codecs following the standard guidelines. The streamer is not part of market applications and its sole purpose is to create simulation data. To this date, it is enabled with variable streaming rate, though no *real* media data are sent.

The acquisition of simulation data is made through a network port, a file or a capture device, depending on the type of session. Media information is then tokenized and packetized in RTP packets. Each connection is represented by a *MMSession* object serving as a stream manager (Figure 7.4).

Each time a connection is requested, an instantiation of a *MMSession* is made. From an object-oriented point of view, it consists of three sub-objects: the *stream controller*, the *packet sender* and the *packet receiver*.

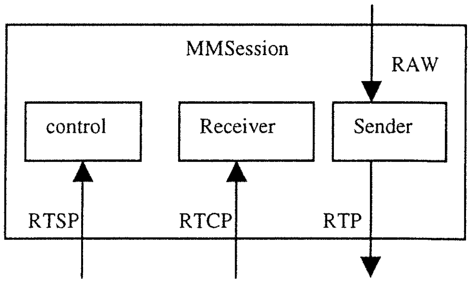


Figure 7.4 : A MMSession Object.

Figure 7.4 shows the relation. The *controller*, as its name implies, controls the state of the stream. It provides stream manipulation functionality that match conventional video operations such as 'start', 'stop' and 'pause'. For the time being, it is not possible to browse through streams with commands like 'forward' or 'rewind' since only a few codecs are capable of such support. Since our goal was to accommodate as many media as possible, it was decided to strictly use mnemonics supported by all A/V compressors. Media specific QoS monitoring is carried in each *MMSession* object by making use of RTCP packets. Throughput, jitter and lost rate are calculated in accordance with algorithms presented in chapter 5.

The *Real Time Streaming Protocol* (RTSP) is now established as a standard in multimedia tools for stream control purposes. Most commercially available products are using this convention. In that sense, a set of mnemonic replicating its most important features was implemented on the reliable TCP links since guarantees of arrival are desired for convenience.

The media information is carried by the *Real Time Protocol* (RTP), which has been specially defined to carry time sensitive data. RTP works closely with its control protocol, the *Real Time Control Protocol* (RTCP), which introduces the needed network feedback information useful for stream adjustment. RTP and RTCP are respectively handled by the *sender* and *receiver* objects, running in two independent threads of execution to reduce delays in both operations.

Since media stream can easily tolerate information losses, the communication links are established over UDP connections. The latter is much faster than its reliable TCP counterpart, a feature that outweighs its lossy characteristic. Moreover, it is now a widely accepted practice to run media streams over UDP, which is another step toward interoperability.

The uncoloured blocks depicted on figure 7.1 are a genuine part of the H.323 stack. On the same figure, it is also possible to distinguish between two kinds of arrows.

The thin black ones indicate that related blocks exchange internal communication messages. Messages can be of any kind, ranging from QoS violation alerts to connection information used by several modules. Communication between blocks is asynchronous and assumed to be reliable.

Most internal communication is triggered by incoming network messages carrying information. This information can be media data, connection related messages or stream control messages. Each incoming message is handled in accordance with the port on which it is received. The carried payload is then sent to different modules for proper manipulation. White thick arrows highlight the exchange of data between stack blocks.

Finally, both client and server work the same way. Most parts are present at both ends of the connection to allow bi-directional QoS communication. This feature is usually required when peers are senders and receivers at the same time, as is the case for on-line teaching.

7.2 – Separation of the Work:

As specified before, our QoS stack in mainly concerned with three aspects: 1) an easier QoS definition and manipulation, 2) a multi-network transport layer for the support of multiple network interface and 3) increased interoperability capabilities by working with the H.323 standard.

All three students working on the project took part in the design of the general architecture of the QoS stack. The definition of the modules along with their capabilities is the result of a joint effort. In a similar way, communication protocol and the nature of the exchanges between components have been established in accordance with the communication requirements of each block.

The implementation task was separated following the aspects presented above. More precisely, Yong Guo undertook the realization of the *QoS manager* along with the QoS definition, specification and management. Jean-Marc Ng Wing Keng implemented the *transport controller* and the related network management pertaining to such matter.

The author concentrated on all H.323 issues such as the generation of streams (*stream generator*), management of connection acceptance processes (*connection manager*), simulation/replication of complex H.323 procedures and compliance with involved protocols. He is also responsible for the production and computation of network feedback information used in the stream adaptation process.

Aside from performing the task of integrating of all the above-mentioned components in a framework environment the author created, designed and implemented a messaging mechanism providing reliable and asynchronous information exchanges between components. Finally, following Guo's QoS representation, the author designed and implemented a client end-application capable of exploiting newly introduced facilities conveyed by the QoS stack.

7.3 – Assumptions & Tradeoffs:

As seen in section 7.1, designing and realizing new QoS related services in the H.323 standard is a tremendous task. First and foremost, it is crucial to have a thorough understanding of the current H.323 model and the interworking constituent technologies. An in-depth knowledge of actual IP challenges in terms of multimedia streaming and how they should be handled is also a definite asset. Finally, a clear and finite definition of the final product and assumptions taken in such a process must be recognized. In this context, the following paragraphs provide a list of assumptions and tradeoffs inherent to the earlier described QoS protocol stack.

Our first assumption was that network equipment is capable of offering packet differentiation through one or a combination of IntServ, DiffServ and MPLS core switching schemes. At end-system stations, we assumed that media acquisition, compression and rendering were provided through software implementation and therefore didn't deal with those issues. We also recognized that media information is most likely to be packaged in RTP by that same software. Since the goal of the suggested stack is to adapt connections' bit rate with network fluctuation, codec implementation should offer dynamic tuning capabilities. Such property is supported by some standards, which have obviously been selected first.

Most architecture supporting real-time services work with operating systems that are enhanced with time-critical mechanisms. These OS are either commercial products or home bred implementations. For complexity, expertise and programmer scarcity reasons, we did not work on those aspects and assumed that resources were always sufficient in both server and client machines.

Nonetheless, we are well aware that such utopian situations will most likely never happen and future work on this project should address those issues. The same remark applies to stream synchronization and reconstruction, two important aspects that were also set aside.

Tradeoffs were introduced to lighten the heavy implementation task. Since this work deals extensively with standard protocols, correct implementation would have taken years to complete. Therefore, we built our own set of protocols that closely imitate the behaviour of standard specifications. This is the case for connection establishment protocols, application capability distribution and QoS data exchange. QoS parameters are closely related to the application and thus, are not relevant for all QoS aware architectures. The latter however, is usually a characteristic common to all projects of this type.

Although convenient to implement, the above assumptions and tradeoffs implies that some features are proprietary and thus not supported by all types of multimedia tool.

7.4 – Summary:

In this chapter, we focused on our own implementation of a QoS aware protocol stack capable of sustaining real-time services for MM application. To do this, we used the knowledge contained in all previously addressed material. The result is an architecture based on the H.323 model enhanced with features allowing common users to specify QoS parameters more efficiently.

We also add significant contributions to the actual standard in terms of adaptation strategies for QoS degradation, group management, per-flow monitoring services and decision making based on user specifications. The model is supported by a transport layer that works seamlessly with multiple commutation technologies such as IP, ATM, and airwaves. The design of a compatible client ensuring bi-directional QoS support has also been completed. Finally, it is important to remember that the *stream generator* and the *connection manager* are modules that have been introduced as conveniences simulating H.323 features and are not part of the standard stack.

We are well aware that added features and utilisation of proprietary protocols induce serious interoperability problems. These design tradeoffs were supported by many constraints that would have otherwise compromised the provability of our concepts.

The next section describes a concreate realization of the architecture presented above. Therein, details about simplified H.323 protocols are given along with the general behavior of the adaptation algorithm. A glimpse about the internals of the multi-network transport layer and some simulation results conclude the work.

8 – Project Implementation:

This chapter is concerned with the concrete realization of the QoS aware protocol stack of chapter 7. We present several aspects and implementation choices that lead to a workable prototype.

More precisely, we give details about the simplified connection process made possible by our home bred set of mnemonics, the general behavior of the adaptation algorithm and the internals of the multi-network transport layer. Before we conclude by showing some of our results, the client and server monitoring interfaces are presented with a small description of the rendered information.

8.1 – Communication Protocol Between Modules:

Many interactions between internal parties of the stack must co-operate when a connection request is logged. To ensure that information is properly transmitted among entities, a suite of internal messages had been defined.

Bellow, we give a sampling of the connection acceptance process and the messages that are exchanged (figure 8.1). The latter depicts a correct execution sequence for a valid session establishment. Error conditions and connection denials are not presented since they only represent a small extension of the process. All message exchanges within the stack are asynchronous. Therefore, once a module sends information to another, it does not wait for an acknowledgement and relies on the native OS's inter-thread message interface to reliably carry the information.

The exchange of messages follows the MSC formalism. The name of the messages are exactly the same as found in the implementation and the syntax is given in appendix 1. The notation is in *C* (the programming language) for conformity and clarity.

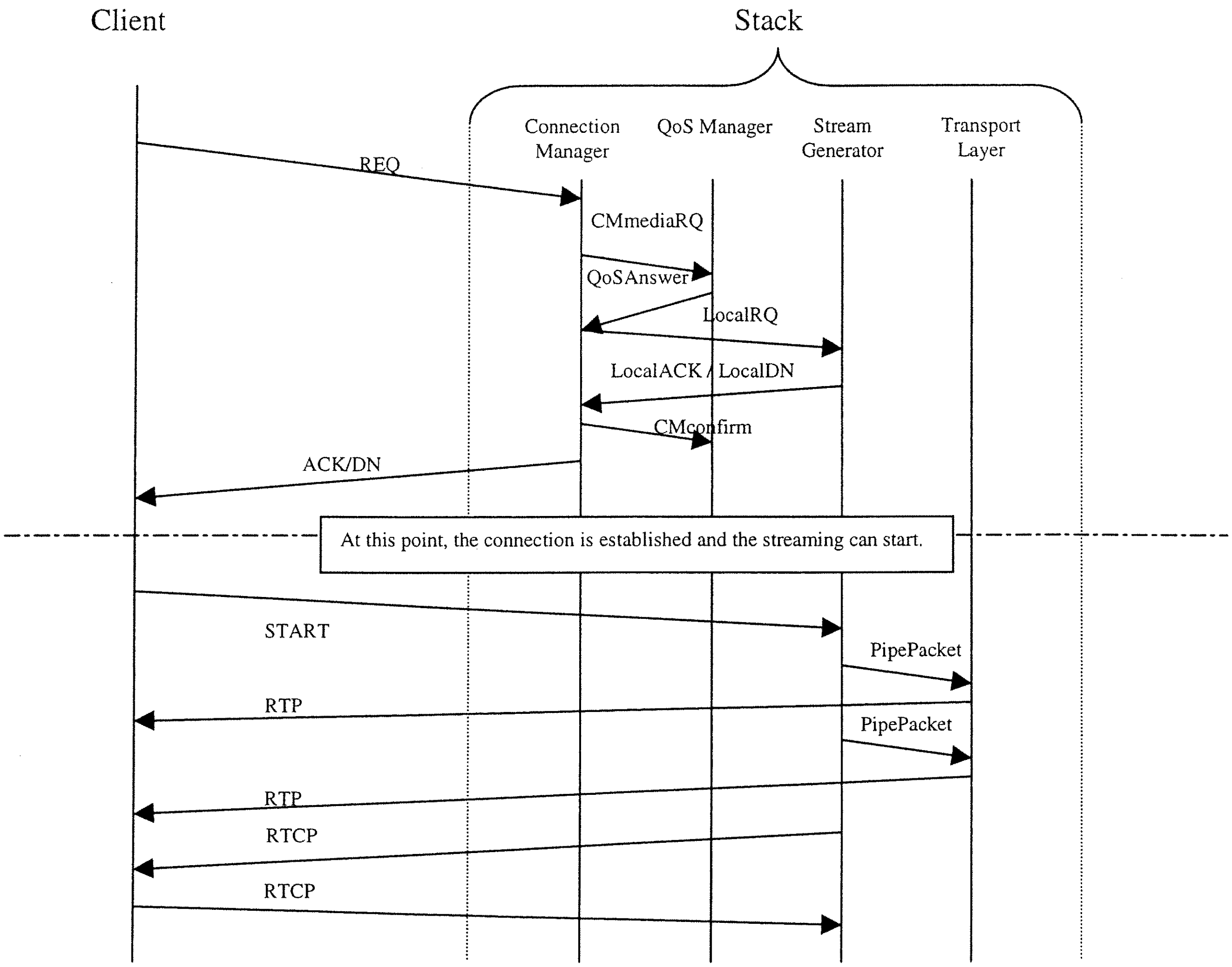


Figure 8.1: A simple connection establishment examples.

The connection process is incremental, meaning that if one of the concerned modules cannot cope with the amount of requested resources, the process automatically ends and the incoming call is dropped.

The *Connection Manager* is the first one involved. It accepts calls from a predefined port and extracts all network related information such as IP addresses and RTP/RTCP port numbers. The conveyed data are stored in a temporary storage and the call is placed on a waiting list. Information about the client and the required QoS is sent to the *QoS Manager* that is responsible for accepting or rejecting the call if internal resources are not sufficient. It is also concerned with creating and managing groups.

If the *QoS Manager* accepts, it sends a notification to the *Connection Manager.* In turn the latter issue an internal media request to the *Stream Generator* that can also accept or reject the request. Upon acceptance, an acknowledgement message is again sent to the *QoS Manager* and a notification dispatched to the client. At this point, session parameters have been established and media streaming can start. This intense message exchange has to be made whenever a connection is requested. It may seems complex, but compared to H.225 and H.245 connection establishment processes, it is rather simple.

The number of messages presented here is only a small subset of all defined messages. The complete list can be found in appendix 1. The amount of messages and the complexity of their structure reflect the seriousness of the design and give insight into the software engineering challenges that had to be addressed.

8.2 – Adaptation Strategies:

Once connections have been established, the QoS aware stack must monitor network changes and adapt the state of the ongoing presentation to model the offered services. The adaptation process is governed by a set of adaptation strategies reflecting users' preferences and capabilities of end-stations.

Adaptation strategies are very difficult to establish since every application has different needs. Moreover, those needs are heavily influenced by the content of a presentation, the environment in which they work and the targeted audience. To this date, no general methodology has been established despite the numerous attempts to define and implement the concept [33,38,89]. Even if the notion of adaptation is changing among authors and their projects, the concept a *degradation path* is commonly encountered. The latter refers to a set of predefined importance/priority (made either by users or ISPs) specifying which QoS parameters are to be affected after a QoS violation occurs, e.g., higher frame rate over image size or sound quality over image quality. The parameters are usually assigned weights that reflect their importance [66].

Initial(QoS)	Initially agreed QoS
New(QoS)	New QoS as result of degradation or enhancement (Present QoS Supported)
ViolationState(T or F)	If New(QoS) < Initial(QoS)
Degrade()	Degrade QoS
Enhance()	Enhance QoS
SwitchATM()	Switch flow to ATM network. Invokes admission control function for ATM connection.
SwitchIP()	Switch flow to IP network
QoSViolation()	QoS Violation notification from Monitoring module
OnIP()	Checking whether on IP network
OnATM()	Checking whether on ATM network
PromptUser("")	Prompt user remark
CheckIP("bandwidth")	Check whether IP network has enough bandwidth to meet QoS requirements.
CheckATM("bandwidth")	Check whether ATM network has enough bandwidth to meet QoS requirements.

Figure 8.2: Summary of the API used in the control of degradation paths [66].

```
If OnIP()  // Assuming the IP network is configured
    If QoSViolation()
        SwitchATM() .or. Degrade() //depending on degradation path profile
        ViolationState(T)
    Else If ViolationState(T) //Already In violation state
        If CheckIP("bandwidth")
            Enhance()
            If initial(QoS) = new(QoS)
                ViolationState(F)
            Else ViolationState(T)
      Else PromptUser("Possibility of upgrade of QoS")

If OnATM()  //Assuming that ATM network is configured
    If QoSViolation()
        Degrade()
        ViolationState(T)
    Else If Violation State(T)
        If CheckIP("bandwidth")// started on IP
          SwitchIP()
          If initial(QoS) = new(QoS)
            ViolationState(F)
          Else ViolationState(T)
        Else If CheckATM("bandwidth")
            Enhance()
            If initial(QoS) = new(QoS)
                ViolationState(F)
            Else ViolationState(T)
```

Figure 8.3: Degradation algorithm for duplicate network configuration [66]. Complexity: *O(n)*.

In figure 8.2, the API specifically designed to control the degradation process is presented. It allows for a high level management of the required information and a lighter programming style for the adaptation algorithm presented in figure 8.3. This algorithm was successfully implemented in the *QoS Manager* and lies at the centre of the decision making process.

In this example, IP and ATM configuration are available, therefore offering more flexibility to intensive steaming needs. It makes extensive use of the network monitoring module that provides information on QoS parameters such as jitter, delay and packet/cell loss rate for video, voice and data flows. The existence of a degradation path profile created by the user to specify the priority of QoS parameters and actions in case of degradation is also taken into account.

8.3 – Multi-Network Transport Layer:

The transport layer of our stack (layer 4 of the OSI standard stack) is somewhat distinctive from all other transport layers encountered to this date. Indeed, it is capable of working seamlessly with several network interfaces. The kind of network is not relevant since all particularities pertaining to its configuration and interaction are embedded in the layer. Only QoS parameters are exchanged and the module makes decision based on the capabilities of its connected network(s). As such and aside from financial considerations, users are not concerned about the above decision process nor the different networks used to exchange media information.

The component modularity shown in figure 8.4 is the key principle enabling easy and clean configuration of supported networks. Time sensitive information is generated and packaged in RTP packets by the *streamer* (section 7). The latter transmits the information to the *Transport Controller* using a pipe, which is a shared memory mechanism built in the operating system allowing for asynchronous information exchanges between processes.

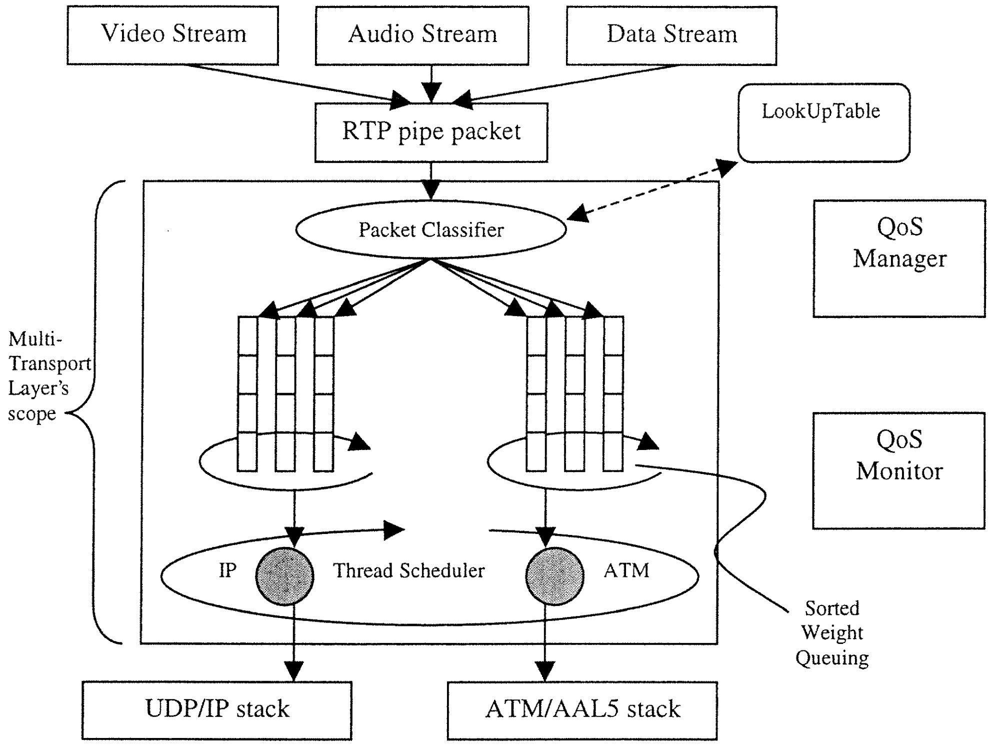


Figure 8.4: The Transport Controller and its Environment.

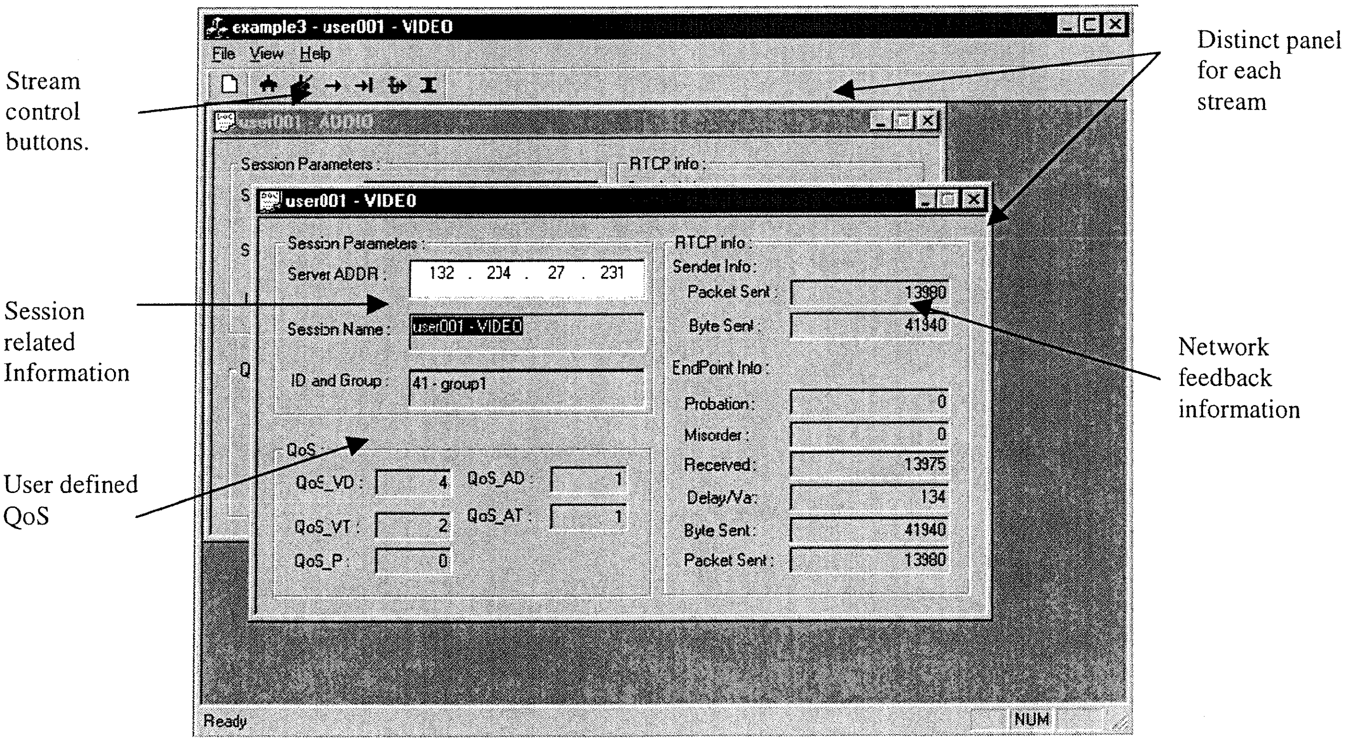
To guarantee processing efficiency, a thread – the *Packet Classifier* – is specifically devoted to the reading of information from the above stated pipe. Upon reception of data packets, the *Packet Classifier* refers to a *LookUpTable* to route the acquired information through different *Priority Queues*. While the *LookUpTable* contains infromation about all ongoing connections such as the network onto which a stream should be sent, *Priority Queues* are temporary information repository gathering incoming packets in accordance with their content and intended network. A set of three queues (audio, video, and data) for each configured network are created and sending priority is based on the weithted fair queuing algorithm of section 3.

In our example, we assume that IP and ATM are configurated. Therefore, selected packets are sent either on UDP/IP sockets or AAL5/ATM cell-interfaces after another thread – the *Scheduler* – performs network access balancement based on availability. Figure 8.4 also shows that the *LookUpTable* is accessed by both the *QoS Manager* and *QoS Monitor* to share media management and control information.

8.4 – Client / Server Interfaces:

The primary concern of this sub-section is the user interface. For that, we'll present and describe the interfaces created at both endpoints. The proposed facilities belong to the streamer and the monitor. Each is updated as network feedback is compiled and rendered to the screen. Also, care was taken in the design of the system to separate screen rendering from internal computation. The interface is therefore not subject to internal blockage caused by network failure and always stays available for user control. The above features and the following material were entirely designed and implemented by the author, as part of his contributions to the project.

The main purpose of the client interface (figure 8.5) is to create and control media sessions. Although not used at this side of the connection, network feedback is displayed for convenience. It gives a good approximation about the provided QoS in network terms. For users that are not familiar with such notation, we also include user-defined QoS parameters. Both information panels are respectively updated upon feedback packet reception and notifications of QoS violations from servers.


example3 - user001 - VIDEO
File View Help
user001 - AUDIO
user001 - VIDEO
Ready
NUM
Stream control buttons.
Distinct panel for each stream
Session related Information
Network feedback information
User defined QoS

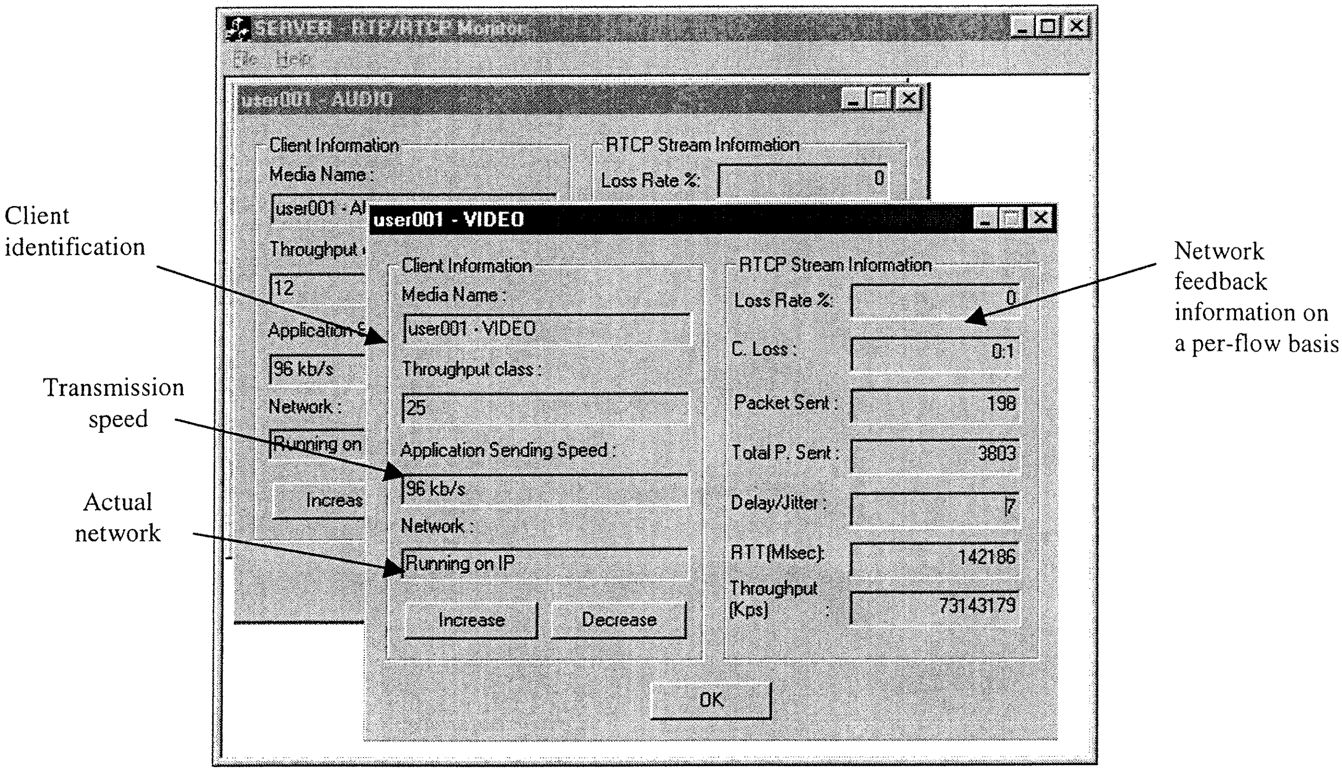
Session Parameters:
Server ADDR: 132 . 2?4 . 27 . 231
Session Name: user001 - VIDEO
ID and Group: 41 - group1

QoS:
QoS_VD: 4
QoS_AD: 1
QoS_VT: 2
QoS_AT: 1
QoS_P: 0

RTCP info:
Sender Info:
Packet Sent: 13980
Byte Sent: 41940
EndPoint Info:
Probation: 0
Misorder: 0
Received: 13975
Delay/Va: 134
Byte Sent: 41940
Packet Sent: 13980

Figure 8.5: Client's main monitor interface.

Functionally speaking, the server side of the connection doesn't need an interface. As with the client side, it was logically separated from internal computing to avoid display blockages. It also runs in its own thread of execution to maximize server performance and avoid overhead. Moreover, since it reflects session evolution, users don't have access to control functionalities. The interface is governed by client commands coming from foreign sites.



user001 - AUDIO

Media Name: user001 - AI

Throughput: 12

Applicatio: 96 kb/s

Network: Running on

Loss Rate %: 0

user001 - VIDEO

Media Name: user001 - VIDEO

Throughput class: 25

Application Sending Speed: 96 kb/s

Network: Running on IP

Loss Rate %: 0

C. Loss: 0.1

Packet Sent: 198

Total P. Sent: 3803

Delay/Jitter: 7

RTT(Mlsec): 142186

Throughput (Kps): 73143179

Client identification

Transmission speed

Actual network

Network feedback information on a per-flow basis

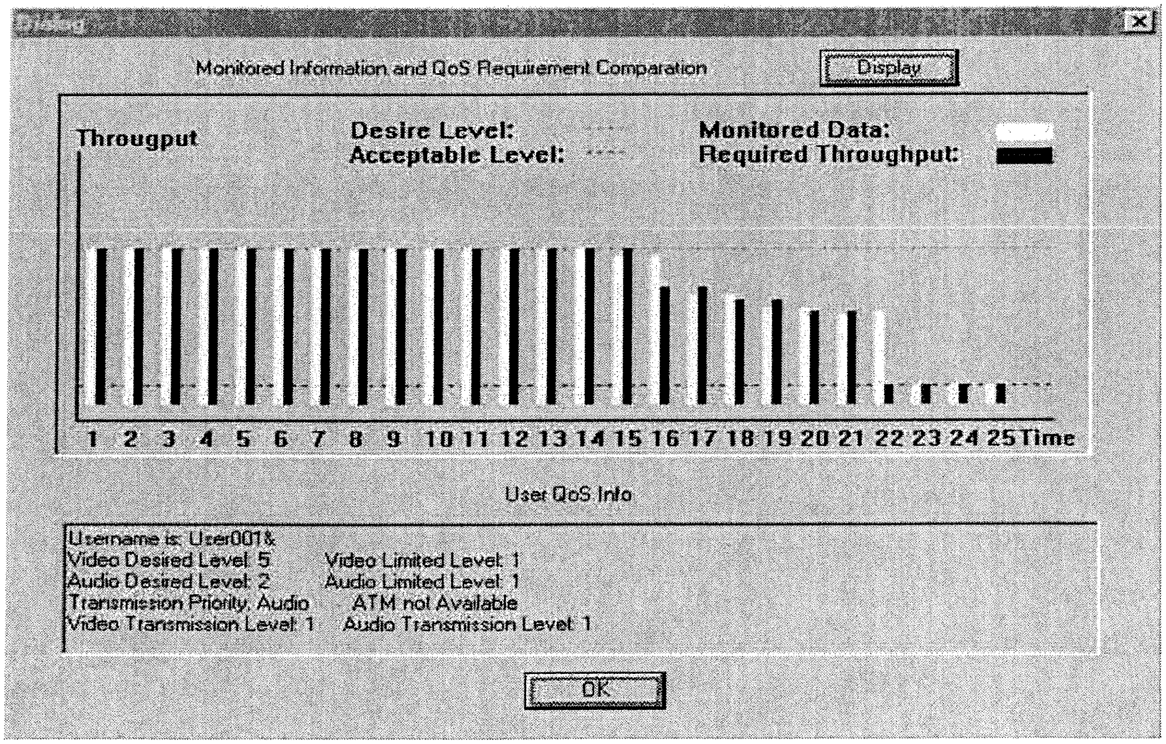
Figure 8.6: The server's monitor interface.

The monitor interface (figure 8.6) at the server side conveys much more information than its counterpart. Aside from network feedback information seen on the right part of the panel, the transmission speed, the network providing the service and the throughput class are presented. As with the client, each panel is concerned with information coming from one media stream whose user name and type are displayed in the first field of the dialogue box. Finally, *Increase* and *decrease* buttons were also added to simulate network quality fluctuations and used to create test-bed events.

8.5 – Simulation Results:

To help visualize simulation results, Yong Guo put tremendous efforts into creating an interface gathering users' QoS specifications, monitoring information and degradation behavior. Figure 8.7 shows such interface and highlights the effectiveness of our degradation algorithm.

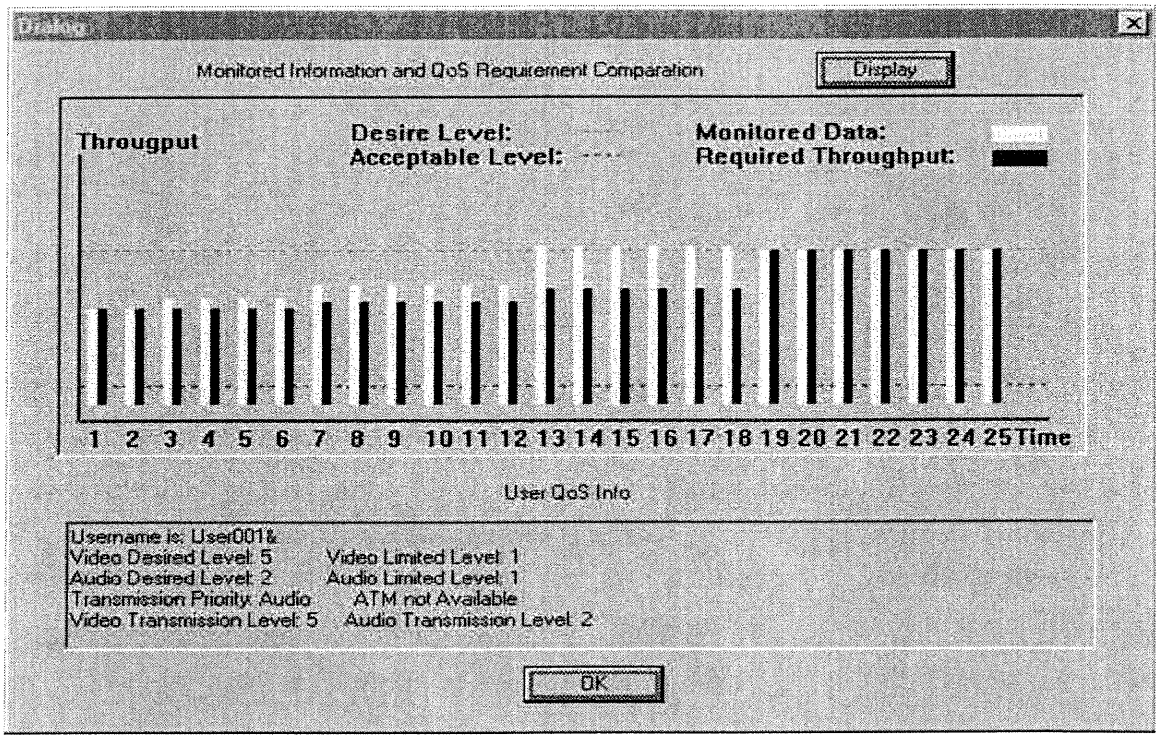
In the upper panel the network throughput is displayed, as conveyed by the last 25 RTCP packets against user define preferences. The dark bands represent the throughput required by media stream at different quality levels. Levels are specified by the user in its *user profile* and dispatched to the server in the connection establishment process. The pale bands reflect the actual network throughput at the reception of network feedback. The results clearly demonstrate that the first 14 packets received premium performance. Service degradation is reported by the 15th reception. This is automatically handled by the *QoS Manager* that instructs a stream rate diminution. The next six packets also indicate that network quality is decreasing and again, stream's bit-rate is adapted to model such events.


Dialog
×
Monitored Information and QoS Requirement Comparation
Display
Througput
Desire Level:
Acceptable Level:
Monitored Data:
Required Throughput:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 Time
User QoS Info
OK

Username is: User001&
Video Desired Level: 5 Video Limited Level: 1
Audio Desired Level: 2 Audio Limited Level: 1
Transmission Priority: Audio ATM not Available
Video Transmission Level: 1 Audio Transmission Level: 1

Figure 8.7: QoS adaptation to a network service decrease.

At the reception of the twenty-second packet, the lowest rate is adopted and stays at that level until network resources become available again. Another choice could have been to start another connection over ATM and have the media flow routed on this new service. The latest solution is left to the user to specify in its requirement and assumes he is ready to pay for such functionality.



```
Username is: User001&
Video Desired Level: 5       Video Limited Level: 1
Audio Desired Level: 2       Audio Limited Level: 1
Transmission Priority: Audio   ATM not Available
Video Transmission Level: 5   Audio Transmission Level: 2
```

Figure 8.8: QoS adaptation to a service increase.

Figure 8.8 shows how our solution reacts to a service increase. It is easy to notice that the *QoS Manager* is much more careful in augmenting the rate than in decreasing it. This is supported by the belief that network services are most likely expected to decrease much faster than they'll increase. In that sense, the algorithm adjusts the bit-rate incrementally, based on positive re-enforcement from network services. If the connection had been routed on another network, the *QoS Manager* will optimize its options and switch it back to the original service provider. The threshold values and decision parameters can be changed dynamically to accommodate a wider range of requirements and offer more flexible services.

8.6 – Implementation Framework:

The simulation results of section 8.5 were acquired from test cases made at different times of the day in our telecomputing laboratory at the University of Montreal. The targeted platforms are PCs equipped with 433Mhz Intel celeron processors and 128 Mb of RAM, working under windowsNT 4.0. 10 Mbs Ethernet network cards are connected to a 100Mbs optic fiber network governed by a central *hub* generally running at 75% of its capacity.

While acknowledging the fact that this environment does not adequately reflect the congestion problems of an open Internet, we believe that the heterogeneity of computers and applications using the network are closely reflect reality. Indeed, the same network is used to interconnect PC, Mac, Sun and HP platforms. The operating systems and services required by each are highly variable and therefore introduce several forms of service impairments usually common to the Internet.

We estimate that the total number of code lines goes beyond 25,000, all programmed in *C/C++* with Microsoft's VC++ 6.0 compiler. We also used MCF interface classes in the design of user interfaces. To synchronize the access of resources and data structures from the average 15 threads working simultaneously, we used guarding mechanisms such as semaphore, mutex and shared resources provided by the native Win32 system library. Finally, all networking interactions were made possible by the Winsock 2.0 library and programmed in *C*.

To help in the support of the H.323 protocol stack, we borrowed a standard compliant product developed by Elemedia Software Inc. [29], a subsidiaries of Lucent Technologies. Despite many collaboration efforts from our side, the lack of support by the former entity prevented us from taking advantage of most features provided by the H.323 standard. We were therefore unable to implement the required support for audio and video streams and had to fill functionality gaps with the aforementioned set of in-house protocols and simulation modules (*stream generator* and *connection manager*).

8.7 – Summary:

Section 8 described some of the important implementation issues of our QoS aware protocol stack. Other protocols viewed in section 5 and 6 were also put to work. The H.323 protocol stack was borrowed from Elemedia Software [29] who produced a standard H.323 compliant protocol stack that includes all the functionality, ranging from call establishment to RTP/RTCP library. For complexity reasons and lack of cooperation from the manufacturer, most features were not exploited. In fact, only the RTP/RTCP library had been used, even though most results were useless since no documentation was provided.

Since the foundation of the architecture could not be used, we implemented our own mechanisms that replicate the behavior of standard H.323 protocols. Such was the case with the call establishment process, although we didn't deal with gatekeepers, MCUs and gateways. Home bred protocols had been designed to model application level QoS specifications. Algorithms capable of translating and working with degradation paths were also developed.

The novel concept of our work is definitively the introduction of a multi-network enabled transport layer. It consists of a series of network interfaces and queues scheduled by a sorted weighted fair queueing algorithm that provides optimum sequencing results, as mentioned in section 3 and proved by many efforts on the topic.

Finally, client and server primary interfaces for monitoring were presented with a short description of the information they report and how it is used. Simulation results close the present section by showing how our degradation algorithm handles network fluctuations.

The next chapter concludes this work by summarizing the aforementioned content. An insight into the remaining challenges to address is also given along with guidelines for future expansions.

9 – Conclusion:

9.1 – Thesis Summary:

The master thesis presented to you addresses the problem of carrying time sensitive information – audio, video, and data – over network infrastructures that don't make provisions about time delivery and service availability. An example of such networks is governed by the IP protocol, whose interconnections are commonly called the Internet.

The Internet is the most common and available network structure in use today. Since its early beginning in the 70's, its relatively low cost, flexibility and reliability characteristics have helped distributed users in many aspects of their work. As the number of connected users became more important and technology advancements significantly increased available bandwidth, new types of information exchanges were requested. Indeed, audio, video and other real-time content such as assembly line data are now transported over IP.

The main problem stemming from these newly introduced media is that IP was not designed for such purposes. From the start, IP was not suited for real-time acquaintance since it is packet based and that TCP – its transport protocol – replicates corrupted or lost packets infinitely if required to. Another disturbing factor is that the Internet is suffering from its own popularity and therefore always congested. Congestion introduces severe delays in core network equipment and leads to packet loss and corruption. Finally, the highly heterogeneous cloud brought by IP network interconnection makes any service prediction impossible.

The above-described problems lead researchers to design and implement solutions that could reduce the effect of IP's real-time deficiencies. Although undoubtedly ingenious, their work has only corrected some of the encountered gaps and many more are still to be addressed to efficiently enable IP networks with real-time capabilities.

Far from introducing a revolutionary model encompassing the actual challenges related to multimedia transport in IP, our work presents a gathering of existing technologies and solutions. The goal is to take as many advantages found in available concepts and models to build a protocol stack capable of sustaining user QoS requirements. The final product should be used for distant learning and work in the Internet environment.

To undertake such a colossal task, we've reviewed state-of-the-art concepts in this document, as well as technologies related to quality of services and multimedia distribution in non-guaranteed environments.

The adventure began with the presentation of actual work in terms of real-time multimedia models (architecture and protocol stack for end-systems). More precisely, MM applications were classified in three categories encompassing the available implementations. Requirements for such products were outlined and the fact that they significantly differ from conventional data transfer applications was recognized. High throughput, restrictive time constraints, service commitment and group communication are all valid examples.

We then followed with basic principles that constitute the foundation and support of QoS in streaming application. In doing so, we saw that packet marking, traffic isolation, maximum utilization of resources and call admission control are widely accepted facts lying at the bottom of successful QoS implementations. To concretize these concepts, enabling algorithms such as the weighted fair queueing and the leaky bucket policing method were explained. Many efforts on the subject proved that the aggregation of both ensures a firm and mathematically provable bound on delays that packets experience in network switching equipment.

Still related to basic QoS principles, management activities – namely QoS management – were addressed. They manipulate stream characteristics to model the state of the network, adapt to service variations by gracefully decreasing the perceived connection quality and deal with service administration such as accounting and billing.

Chapter 4 describes the essence of existing multimedia applications developed by leading universities renowned for their work in the support of time sensitive flows over network links. The most successful projects are presented along with an assessment of their pros, cons and feasibility.

Chapter 5 presented three protocols actually used by applications requiring media streaming and manipulation. RTP and RTCP have been designed to bring a common agreement on packet format tailored to the transport of time-sensitive information. As such, RTP provides sequence numbers, timestamps, payload identification, and synchronization information believed to have a common use in all stream transport protocol. RTCP brings the network feedback feature not supported by IP. With the conveyed reception report fields, it is possible to have network sampling measures such as the number of packet loss, an approximation of the throughput and an evaluation of the jitters introduced by hops on the data path. As for the manipulation of media streams, RTSP a common choice today. All three protocols are standards, which should allow interoperability among applications.

The above material was combined with the sixth chapter to present the H.323 protocol umbrella. H.323 is a set of architecture components and protocols working together to establish a framework viable for real-time data streaming. It addresses problems pertaining to time constraints and network deficiencies by providing a means for client applications to select parameters that are best suited to their environment. It also delivers a standard set of technology requirements (codec, protocol, signalling) that ensures a common understanding among all participants. H.323 is a standard ITU specification and already up to its second version. To this date, it is the most complete model available. It is backed by several industry players and used in many products ranging from desktop computers to hand held devices.

The H.323 standard is the foundation of our QoS aware protocol stack – chapter 7. As such, our desire was to incorporate its benefits while introducing features that are actually not supported. Three main characteristics had been envisioned. First, an easier QoS representation and specification process to allow for simpler manipulation by unacquainted users. Second, we recognized the fact that media sessions should be capable of adapting to network service fluctuations in accordance with users' specifications. This contrasts with actual implementations where media streams are becoming glitchy with service impairments. Finally, the novel idea of interacting with more than a single transport layer sounded interesting for offering a wider range of possibilities.

Implementing the idea presented in the above paragraph was an endeavouring task. Our first obstacle was the huge amount of interworking technologies involved in the design. The best software engineering techniques alone could not solve all related issues, which left many of them unaddressed. The decision to take an H.323 stack as a foundation was definitively sound since it follows the actual wave. Nevertheless, working with this powerful specification introduced a fair amount of complexity that wasn't been foreseen at first. The lack of support by the manufacturer was also a disruptive factor.

Our second challenge was to merge the concept of quality of service with distant learning application. The former is definitively troublesome to define since it changes from author to author, depending on the research project and target goals. As for learning applications, they can be classified in the "cooperative work tool" class of section 2. Most people will agree that such applications pose severe design and implementation problems induced by the amount of requirements to satisfy – this project was no exception.

Despite these challenges, a working prototype was implemented, proving that our work is viable. To this date, all stack related issues have been addressed and QoS parameters can be sustained for packets carrying data. Audio and video streams are not currently supported since they require supplementary knowledge and modules going beyond the scope of this work. Chapter 8 described our implementation in greater details.

The project described in this thesis was implemented by a joint effort from the author and two other students – Jean-Marc Ng Wing Keng and Yong Guo. Although each participant contributed to all design aspects of the work, Jean-Marc Ng Wing Keng and Yong Guo were respectively concerned with implementing the multi-transport layer and the QoS definition and management. The author's contributions mostly relate to handling or imitating H.323 services and are specifically outlined in the following:

- All H.323-related aspects such as protocol definition, media streaming and connection specification, establishment and management.
- Design and implementation of a client module capable of working with the server prototype.
- Implementation, generation and management of RTP/RTCP simulation packets along with the supporting *stream generator* module.
- Production of QoS information based on the reception of RTCP packets.
- Design and implementation of a set of protocols imitating connection-specific H.323 protocols used in the *connection manager*.
- Design and implementation of a global framework into which modules from other collaborators were inserted.
- Design and implementation of a messaging API allowing reliable, asynchronous message exchanges between architecture components.
- Design of the state machine allowing for the introduction, manipulation and departure of users in multimedia session.

9.2 – Future Work:

Our final statement concerns some of the work that should be undertaken to bring the actual realization to a fully working product. The intention is to guide other researchers in their decisions and provide some guidelines that may ease the design process of similar prototypes. All topics are considered to have an equal value and the order of presentation is not related their importance.

- An H.323 protocol stack is a definite asset. When making your selection, make sure that you have sufficient knowledge and that the source code is available.
- RTP/RTCP protocols are mandatory in the actual design of streaming applications. Therefore, an API capable of packaging information according to the standard is needed.
- The base operating system definitively needs to be enhanced with real-time characteristics; this is not a strong asset in window-based products.
- The design of media codec is required for media compression. Several are freely available, but modifications must be made to have them work with other stack modules.
- If working with more than one network, packet synchronization issues must be addressed to reconstruct media flows once received at the other end of the channel.
- Other protocols such as SIP and SDP should be envisioned to replace the expensive H.323 stack.

References:

[01] Anderson, D. *and al.*, "A File System for Continuous Media", ACM Transactions on Computer Systems, Vol. 10, No. 4, 1992.

[02] Andersson, L. *and al.*, "Label Distribution Protocol", Internet draft, draft-ietf-mpls-ldp-02.txt, 1998.

[03] Aurrecoechea, C., Campbell, A., and Eleftheriadis, A., "Meeting QoS Challenges for Scalable Video Flows in Multimedia Networking", Center for Telecommunication Research, Columbia University, New York, USA, 1995.

[04] Aurrecoechea, C., Campbell, A.T., and Hauw, L., "A Survey of QoS Architectures", Center for Telecommunication Research, Columbia University, New York, USA, 1998.

[05] Banerjea, A. *and al.*, "The Tenet Real-Time Protocol Suite: Design, Implementation, and Experiences", Technical Report 94-059, International Computer Science Insitute, Berkeley, USA, 1994.

[06] Blake, S. *and al.*, RFC 2475, "An Architecture for Differentiated Services", 1998.

[07] Bochmann, G. and Hafid, A., "Some Principles for Quality of Service Management", Distributed Systems Engineering Journal, Vol. 4, No. 1, 1997.

[08] Boerjan, J., *and al.*, "The OSI 95 Transport Service and the New Environment", Internal Report No. MPG-92-38, Department of Computing, Lancaster University, Lancaster, England, 1992.

[09] Braden, R., Clark, D., and Shenker, S., RFC 1633, "Integrated Services in the Internet Architecture: an Overview", 1994.

[10] Braden, R. *and al.*, RFC 2205, "Resource ReserVation Protocol – Version 1, Functional Specification", 1997.

[11] Budd, T. A., "Classic data structures using C++", Addison-Wesley, San Jose, USA, 1994.

[12] Busse, I., Deffner, B., and Schulzrinne, H., "Dynamic QoS Control of Multimedia Application based on RTP", First International Workshop on High Speed Networks and Open Distributed Platforms, St. Petersburg, Russia, 1995.

[13] Cadzow, J.A., "Foundation of Digital Signal Processing and Data Analysis", MacMillan, New York, USA, 1987.

[14] Callon, R. *and al.*, "A Framework for Multiprotocol Label Switching", Internet draft, draft-ietf-mpls-framework-05.txt, 1999.

[15] Campbell, A. *and al.*, "A Continuous Media Transport and Orchrestration Service", Proceedings of ACM SIGCOMM'92, Maryland, USA, 1992.

[16] Campbell, A., Coulson, G., and Hutchinson, D., "A Multimedia Enhanced Transport Service in a Quality of Service Architecture", Proceeding of the Fourth International Workshop on Network and Operating System Support for Digital Audio and Video, Lancaster Universtiy, Lancaster, England, 1993.

[17] Campbell, A., Coulson, G., and Hutchinson, D., "A Quality of Service Architecture", ACM Communications Review, Vol. 24, No. 2, 1994.

[18] Cho, S. and Shin, Y., "Multimedia Service Interworking over Heterogeneous Networking Environments", IEEE Network, Vol. 13, No. 2, 1999.

[19] Cisco Systems Inc., "Congestion Management Overview", http://www.cisco.com/univercd/cc/td/doc/product/software/ios120/12cgcr/qos_c/qcpart2/qcconman.htm , 1999.

[20] Clark, D.D., Shenker, S., and Zhang, L., "Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism", Proceeding of ACM SIGCOMM'92, USA, 1992.

[21] CMPT 365 – Multimedia Systems, http://www.cs.sfu.ca/CourseCentral/365/li/index.html .

[22] Columbia University, "rtptools: RTP implementation library", ftp://ftp.cs.columbia.edu/pub/schulzrinner/rtptools/ .

[23] Coulson, G., and Blair, G.S., "Micro-kernel Support for Continuous Media in Distributed Systems", Computer Networks and ISDN Systems, Special Issue on Multimedia, No. 26, 1993.

[24] DataBeam Corporation, "A Primer on the H.323 Series Standard, Version 2.0", http://www.databeam.com, 1998.

[25] Delgrossi, L. *and al.*, "Media Scaling for Audiovisual Communication with the Heidelberg Transport System", Proceeding of ACM Multimedia '93, Ahaheim, USA, 1993.

[26] Demers, A., Keshav, S., and Shenker, S., "Analysis and Simulation of a Fair Queueing Algorithm, Internetworking: Research and Experience", Vol. 1, No. 1, 1990.

[27] Dermler, G. *and al.*, "Constructing a Distributed Multimedia Joint Viewing and Tele-Operation Service for Heterogeneous Workstation Environment", Proceedings of the IEEE Workshop on Future Trends of Distributed Computing Systems, Lisboa, Portugal, 1993.

[28] El-Marakby, R. and Hutchinson, D., "Delivery of Real-time Continuous Media over the Internet", Proceedings of the 2nd IEEE Symposium on Computers and Communications (ISCC'97), Egypt, 1997.

[29] Elemedia Software inc., http://www.elemedia.com .

[30] Eriksson, H., "Mbone: The multicast backbone", Communication of the ACM, Vol. 37, No. 8, 1994.

[31] Ferrari, D. and Verma, D., "A Scheme for Real-Time Channel Establishment in Wide-Area Networks", IEEE JSAC, Vol. 8, No. 3, 1990.

[32] Ferrari, D. *and al.*, "Network Support for Multimedia", Technical Report 92-072, International Computer Science Insitute, Berkeley, USA, 1992.

[33] Gecsei, J., "Adaptation in Distributed Multimedia Systems", IEEE Multimedia, Vol. 4, No. 2, 1997.

[34] Gonzalez, R., and Woods, R., "Digital Image Processing", Addison Wesley, San Jose, USA, 1992.

[35] Guérin, R. and Peris, V., "Quality-of-service in Packet Networks: Basic Mechanisms and Directions", Computer Networks, Vol. 13, No. 3, 1999.

[36] Gutekunst, T. *and al.*, "A Distributed Multimedia Joint Viewing and Tele-Operation Service for Heterogeneous Workstation Environments", In Proceedings of the Workshop on Distributed Multimedia Systems, Stuttgart, Germany, 1993.

[37] Hafid, A. "Gestion de la Qualité de Service dans les Applications Multimédia", Ph.D Thesis, University of Montreal, Montreal, Canada, 1996.

[38] Hafid, A. and Bochmann, G.V., "Quality of Service Adaptation in Distributed Multimedia Applications", Multimedia Systems Journal, Vol. 6, No. 5, 1998.

[39] Hafid, A. *and al.*, "Distributed Multimedia Application and Quality of Service: A Review", The Electronic Journal on Networks and Distributed Processing, issue 6, 1998.

[40] Hehmann, D.B. *and al.*, "Implementing HeiTS: Architecture and Implementation Strategy of the Heidelberg High Speed Transport System", Second International Workshop on Network and Operating System Support for Digital Audio and Video, IBM ENC, Germany, 1991.

[41] Heinanen, J. *and al.*, "Assured Forwarding PHB Group", Internet draft, draft-ietf-diffserv-af-04.txt, 1999.

[42] Huard, J-F. *and al.*, "Meeting QoS Guarantees by End-to-End QoS Monitoring and Adaptation", Workshop on Multimedia and Collaborative Environments of performance in Distributed Computing, Syracuse, USA, 1996.

[43] IETF-MPLS working group, http://www.ietf.org/html.charters/mpls-charter.html.

[44] IETF, RFC 2250, "RTP Payload Format for MPEG1/MPEG2 Video", 1998.

[45] IETF-RSVP working group, http://www.ietf.cnri.reston.va.us/html.charters/rsvp-charter.html .

[46] ITU-T Recommencation H.261, "Video Codec for Audiovisual Services at p X 64 kb/s", 1993.

[47] ITU-T Recommendation H.323, "Packet Based Multimedia Communication Systems", 1998.

[48] Jacobson, V. *and al.*, "An Expedited Forwarding PHB", Internet draft, draft-ietf-diffserv-phb-ef-02.txt, 1999.

[49] Kerherve, K. *and al.*, "On Distributed Multimedia Presentational Applications: Functional and Computational Architecture and QoS Negotiation", Proceedings of High Speed Networks Conference, Vancouver, Canada, 1994.

[50] Kibrick, R. *and al.*, "Collaborative Remote Observing with the W.M. Keck Observatory", ACM Interactions, Vol. 3, 1998.

[51] Klett, S., "Mbone: Videoconferencing over Internet", CROSSCUT, Vol. 5, No. 1, 1996.

[52] Knightly, E. W. and Shroff, N. B., "Admission Control for Stastistical QoS: Theory and Practice", IEEE Network, Vol. 13, No. 2, 1999.

[53] Kurose, J. and Ross, K., "Computer Networking and Internet Protocols", Addison-Wesley, San Jose, USA, 1999.

[54] Lazar, A.A., "A real-time management, control and information transport architecture for broadband networks", Proceeding of the Int'l Zurich Seminar on Digital Communication, Switzerland, 1992.

[55] Lazar, A.A., "Challenges in Multimedia Networking", Proceedings of the International Hi-Tech Forum, Osaka'94, Japan, 1994.

[56] Liou, M., "Overview of the p*64 kbits/s video coding standard", Communication of the ACM, Vol. 34, No. 4, 1991.

[57] Liu, C. and Layland, J., "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment". ACM Journal, Vol. 20, No. 1, 1973.

[58] Liu, C., "Multimedia Over IP: RSVP, RTP, RTCP, RSVP", Internal Report, University of Ohio State, http://www.cis.ohio-state.edu/~jain/cis788-97/ip_multimedia/index.htm.

[59] Lougher, P., "The Design of a Storage Server for Continuous Media", Ph.D Thesis, Lancaster University, Lancaster, England, 1993.

[60] McCanne, S. and Jacobson, V., "Vic: A Flexible Framework for Packet Video", Proceeding of ACM Multimedia, San Francisco, USA, 1995.

[61] Microsoft Corporation, http://www.microsoft.com .

[62] Minoli, D., "Video Dialtone Technology", McGraw-Hill, 1995.

[63] Mills, D., RFC 1305, "Network Time Protocol Version 3", 1992.

[64] MPEG, MPEG2, MPEG4, H.261, H.263. http://www.stud.ee.etht.ch/~rggrandi/intro.html.

[65] Network Research Group and Lawrence Berkeley National Laboratory, "Vat, An Audio Video Conferencing Application", http://www-nrg.ee.lbl.gov/vat .

[66] Ng Wing Keng, J-M., and Guoy, Y., "Adaptation Mechanisms and Policies", Technical Report, University of Montreal, Montreal, Canada, 1999.

[67] Nicols, K. *and al.*, RFC 2474, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", 1998.

[68] Olson, G.M. *and al.*, "The Upper Atmospheric Research Collaboratory", ACM Interactions, Vol. 3, 1998.

[69] Pancha, P. and El Zarki, M., "MPEG Coding for Variable Bit-rate Video Transmission", IEEE Communication Magazine, Vol. 32, No. 5, 1994.

[70] Parekh, A. and Gallager, R., "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks : the Single-node Case", IEEE/ACM Transactions on Networking, Vol.1, No. 3, 1993.

[71] RealNetworks, http://www.realnetworks.com .

[72] Rosen, E., Viswanathan, A., and Callon, R., "Multiprotocol Label Switching Architecture", Internet draft, draft-ietf-mpls-arch-06.txt, 1999.

[73] Rosenberg, J. and Schulzrinne, H., "Timer Reconsideration for Enhanced RTP Scalability", Proceeding of IEEE INFOCOM, San Francisco, USA, 1998.

[74] RTSP Resource Center, http://www.real.com/devzone/library/fireprot/rtsp .

[75] Schulzrinne, H., "Internet Services: From Electronic Mail to Real-Time Multimedia", KIVS'95, Chemnitz, Germany, 1995.

[76] Schulzrinne, H. *and al.*, RFC 1889, "RTP: A Transport Protocol for Real-Time Applications", 1996.

[77] Schulzrinne, H. and the Audio-Video Transport Working Group, RFC 1890, "RTP Profile for Audio and Video Conferences with Minimal Control", 1996.

[78] Schulzrinne, H., Rao, A., Lanphier, R., RFC 2326, "Real-Time Streaming Protocol", 1998.

[79] Schulzrinne, H., "RTP site at Columbia University", http://www.cs.columbia.edu/~hgs/rtp, 1998.

[80] Schulzrinne, H. and Rosenberg, J., "The IETF Internet Telephony Architecture and Protocols", IEEE Network, Vol. 13, No. 3, 1999.

[81] Shenker, S., Partridge, C., and Guerin, R., RFC 2212, "Specification of Guaranteed Quality of Services", 1997.

[82] Steinmetz, R. and Nahrstedt, K., "Multimedia: Computing, Communications and Applications", Prentice-Hall, 1995.

[83] Tanenbaum, A., "Réseaux", Prentice Hall, 1996.

[84] The Tenet Group, University of California at Berkeley, www.cs.berkeley.edu/Research/Projects/tenet/html/tenet.html .

[85] The IETF Differentiated Services Working Group homepage, http://www.ietf.org/html.charters/diffserv-charter.html .

[86] Toga, J. and ElGebaly, H., "Demystifying Multimedia Conferencing Over the Internet Using the H.323 Set of Standards", Intel Technology Journal Q2'98, http://www.intel.com.tw/technology/itj/q21998/articles/art_4.htm , 1998.

[87] Tryfonas, C., "MPEG-2 Transport over ATM networks", Master thesis, University of California, Santa Cruz, USA, 1996.

[88] Turletti, T. and Huitema, C., "Videoconferencing on the Internet", IEEE/ACM Transaction on Networking, Vol. 4, No. 3, 1996.

[89] Verma, D.C., Zhang, H. and Ferrari, D., "Delay Jitter Control for Real-Time Communication in a Packet-Switching Network", Proceeding of TriComm'91, 1991.

[90] VocalTec, Internet Phone 4.0, http://www.vocaltec.com .

[91] Vogel, A. *and al.*, "Quality of Service Management: a survey", IEEE Journal of Multimedia Systems, Vol. 2, No. 2, 1995.

[92] Vogt, C. *and al.*, "HeiRat – Quality of Service in Multimedia Computer Operating Systems", Multimedia Systems Journal, 1996.

[93] Web ProForum Tutorial: H.323, http://www.webproforum.com/h323/index.html.

[94] Weeks, A., "Fundamentals of Electronic Image Processing", Spie/IEEE Series on Imaging Science & Engineering, 1998.

[95] Wolfinger, B. and Moran, M., "A continuous Media Data Transport Service and Protocol for Real-Time Communication in High Speed Networks", Proceedings of the second International Workshop on Network and Operating System Support for Digital Audio and Video, Germany, 1991.

[96] Wroclawski, J., RFC 2210, "The use of RSVP with IETF Integrated Services", 1997.

[97] Wroclawski, J., RFC 2211, "Specification of the Controlled-Load Network Element Services", 1997.

[98] Xiao, X. and Ni., M. L., "Internet QoS: A Big Picture", IEEE Network, Vol. 13, No. 2, 1999.

[99] Zhang, L. *and al.*, "A Framework for Use of RSVP with Diff-serv Networks", Internet Draft, draft-ietf-diffserv-rsvp-01.txt , 1998.

Appendix 1 – Part 1:

```
/* allHeader.h */

#ifndef allHeader_h
#define allHeader_h

  #include <stdio.h>
  #include <process.h>
  #include <stdlib.h>
  #include <string.h>
  #include <time.h>
  #include <iostream.h>
  #include <math.h>

  #include "stdafx.h"

  #include "rtp/rtp.h"
  #include "globalHeader.h"

  #include <exception>
  using namespace std;

#endif
```

Appendix 1 – Part 2:

```
/* glogalHeader.h */

#ifndef _globalHeader_h
#define _globalHeader_h
#include "winsock.h"

//DO NOT MODIFY.  THESE ARE USED BY THE MESSAGEQUEUE.

#define USER_MESSAGE_ID (WM_USER + 1000)
#define MQ_INTERNAL_ID  (WM_USER + 100)

#define CM_MSG    (WM_USER + 1)  //Message coming from
connectionManager.
#define QoS_MSG   (WM_USER + 2)  //Message coming from QoSManager.
#define QU_MSG    (WM_USER + 3)  //Message coming from QueuModule.
#define ST_MSG    (WM_USER + 4)  //Message coming from streamer/monitor.
#define PG_MSG    (WM_USER + 5)  //Message coming from pinger.



//////Global Mutex guarding the LookUpTable structure /////
static HANDLE LookUpMutex = CreateMutex(NULL,FALSE,"LookUpMutex");
////////////////////


/*********************************************
  Message identifiers used by messages queues
  to differentiate message senders.
*********************************************/

enum{

    // Message sent from connectionManager
      CM_MEDIA_RQ,     //to QoS and ST  1
      CM_RQ,           //to streamer
      CM_CONFIRM,      //to QoS         2
      CM_DENIAL,       //to QoS

      // Message sent from streamer.
      ST_ANSWER,       //to CM          3
      ST_DENIAL,       //to CM
      ST_CALL_CLOSED,

    // Messages sent from QoSManager
      QoS_ANSWER,      //to CM          4
      QoS_GET_TH,      //to PG          5

    // Message sent from Monitor
      MO_UPDATE,       //to QoS         6

      // Message sent from Pinger.
      PG_ASW_TH,       //to QoS         7

      // For UI.
      UPDATE,
      QoS_CHANGE,
      QoS_CHANGE_ST,
      MO_UPDATE_Q,

      // group management related.
      GROUP_COMMAND,      // from streamer to GManager
      GROUP_LIST          // GManager to streamer.
};

// QoS parameter structure
typedef struct{
  int QoS_VD;
  int QoS_VT;
  int QoS_AD;
  int QoS_AT;
  int QoS_P;
}QoS;


/*********************************************
  Messge structure tailored with the
  above identifiers
*********************************************/

typedef struct{
  int connectionID;
  int throughput;
  int reseau;
}ChangeStruct;

//Structure is exchanged between streamer and queueModule
typedef struct{
  int connectionID;
  int packetLength;
  char buffer[2000];
}PipePacket;

//Parameters sent from CM to QoS_M prior to call establishment
typedef struct{
  int localID;              //only for internal message reference.
  char name[9];             //name of the client.
  char ipAddr[256];         //IP address of the client.
  int  video;               // 0 = not video.
  unsigned short rtpPort;   //rtp port to be used by Queue.
  unsigned short tcpPort;   //tcp port that should listen for
QoSViolation.
  QoS qos;
  int groupID;
}CMmediaRQ;




typedef struct{
  int connectionID;
  char name[8];
  int Media;
  QoS  qos;
  char clientIP[256];
  int Network;
  unsigned short tcpPort;
  unsigned short udpPort;
  SOCKET socket;
  int throughput;
}LookUpNode;

//Request response sent to QoS_M to CM.
typedef struct{
  int localID;                   //only for internal message reference.
  int response;            // 0 = denied, 1 = accepted.
  QoS qos;                 // the agreed QoS parameters.
  char reason[256];        // why the connection was denied, if this is
the case.
}QoSAnswer;

typedef struct{
  int throught;
  int delay;         //Average packet transmission time
  int jitter;        //Average difference between each packet transmission
time
  int Loss_Rate;     // From the time of last report till now
}QoSFeedback;

typedef struct{
  int localID;
  char ipAddr[256];
}QoSGetTH;

typedef struct{
  char ipAddr[256];
  int localID;
  char username[8];
  int throughput;
  int delay;
}PgAsw;

typedef struct{
  int localID;
  int throughput;
}PgAswTH;

typedef struct{
 int connectionID;
 int video;        // 0 = not video.
 char name[9];     // name of the client.
}CMconfirm;

typedef struct{
 int video;
 char name[9];
}CMdenial;
typedef struct{
 int connectionID;
 QoSFeedback feedback;
}MOpdate;

typedef struct{
 int connectionID;
 int command;
}STR_GROUP_COMMAND;
```

```
typedef struct{
 int command;
 int *groupList;
}STR_GROUP_LIST;


// DO NOT EDIT.
typedef struct{
   int message;
   int wParam;
   int lParam;
 }USER_MESSAGE;

/*********************************************
  Varibles for queue management
*********************************************/
const int NumPackets = 1; //Number of tested packets
const int NumQueuesIP = 3;   //Number of Queues
const int NumQueuesATM = 3;
const int NumThreads = 3;


#endif
```

Appendix 1 – Part 3:

```
/* myNetworkLib.h */

#ifndef myNetworkLib_h
#define myNetworkLib_h


#include "allHeader.h"

/********************************************
  Internal monitor message identifiers
********************************************/

  #define UI_MSG (WM_USER + 10)
  #define MM_MSG (WM_USER + 11)
  #define TH_MSG (WM_USER + 12)

  #define RESUME_MESSAGE  (WM_USER + 13)
  #define START_MESSAGE   (WM_USER + 14)
  #define STOP_MESSAGE    (WM_USER + 15)
  #define SUSPEND_MESSAGE (WM_USER + 16)

  #define GET_LIST_ITEM   (WM_USER + 17)
  #define REFRESH         (WM_USER +18)

enum{

      //Messages sent from UI
      UI_PLACE_CALL,
      UI_DROP_CALL,
      UI_START_STREAM,
      UI_SUSPEND_STREAM,
      UI_RESUME_STREAM,
      UI_LIST_CALLS,

      //Messages sent from MMSession
      MM_ALL_CALL,
      MM_SINGLE_CALL,
      MM_CLOSED,

      //Message sent from RTP_RTCPListener
      RTP_UPDATE,

      //Message connection establishment
      TH_ANSWER,
      CM_ACK,
      CM_DN
};

typedef struct{
  char ip[256];
  char name[256];
}UIPlaceCall;


/********************************************
  Constant definitions for
  various utilities
```

```
********************************************/

  #define MAX_PACKET_SIZE 4096
  #define MAX_PAYLOAD_SIZE  1024
  #define MAX_PAYLOAD_SIZE 3

  #define STRD_SERVER_PORT 14001
  #define STRD_CLIENT_PORT 14000

  #define SERVER_ADDR "127.0.0.1"
  #define CLIENT_ADDR "127.0.0.1"

  #define DUMMY_INFO "DUMMY_INFO"
  #define DUM "OOO"  //this is 96 bits = char[3].

//DEBUGGING
  #define QUEUE
  #define LOCAL         //enables loop back.
  #define VERBAL        //enables printing of status messages.
  #define VERBAL_STATS   //enables printing of rtcp stats.
  #define VERBAL_ERROR  //enables printing in socket_error.
  #define INTERVAL 60 //inteval between sending two rtcp packets.
// END DEBUGGING

//Identify the state of a MMSession.
  typedef enum{CREATED,STREAMING,SUSPENDED,CLOSED} STATUS;



/*********************************************
   Command packet specification.
   Used in the ConnectionManager class
********************************************/

  typedef enum{_RQ,_ACK,_DN} PACKET_TYPE;

  typedef struct{
   char clientAdd[256];
   unsigned short tcpPort;
   unsigned short rtpPort;
   unsigned short QoSPort;
  }IP_PARAM;

  typedef struct{
   char sessionName[8];
   int  mediaType;
   int  sampling;
  }MEDIA;







  typedef struct{
   int ID;
   IP_PARAM ip_param;
   QoS      qos;
   MEDIA    media;
   int      groupID;
   int      Network;
  }RQ;

  typedef struct{
   int ID;
   IP_PARAM ip_param;
   QoS qos;
  }ACK;

  typedef struct{
   int ID;
   char message[256];
  }DN;


 typedef struct{
   PACKET_TYPE type;

   union{
         RQ request;
       ACK ack;
       DN denial;
   }CONTENT;

 }PACKET;
//  End command packet specification.


 typedef struct{
      int  ID;
      STATUS status;
    char *ip_add;
      unsigned short tcpl;
      unsigned short tcpd;
      char *sessionName;
 }SessionInfo;

 typedef struct{
  int localID;
  RQ  rq;
 }LocalRQ;

 typedef struct{
  char ip[256];
  unsigned short port;
  RQ rq;
 }LocalClnRQ;




 typedef struct{
  int localID;
  int connectionID;
  ACK ack;
 }LocalACK;

 typedef struct{
  int localID;
  DN  denial;
  }LocalDN;

  typedef struct{
   int ID;
   RTCPSenderInfo senderInfo;
   RTPEndpointStats res;
   RTCPReceptionReport rr;
   float rtt;
   float jitter;
   float KbitSec;
   char realName[40];
   char email[40];
   char location[20];
   char tool[20];
  }STR_UPDATE;



//  Utility functions.
//  Used throughout the code.
   int  recvN     (SOCKET s,char *buf,int len,int flag = 0);
   int  sendN     (SOCKET s,char *buf,int len,int flag = 0);
   void setLinger (SOCKET s,short time);
   void getClientInfo(struct sockaddr_in add);
   void show_message(char *message);
   void show_error(char *message);
   void socket_error(SOCKET s, char *message);

   float getRTT(unsigned int ntp_sec,unsigned int ntp_frac,
                unsigned int delay,  unsigned int sent);


   float getNTP64time(unsigned int sec,unsigned int frac);
   float getNTP32time(unsigned int time);
//  End utility functions.

// Timer intervals.
    #define WAIT 1          // Amount of time stream sockets
                            // are allowed to linger before closing.
    #define WAIT_COMMAND 1 // Amount of time a client is willing to
                            // wait after connection request.
// End timer intervals.

   #define MAX_CALL_NUMBER 12

   #define MASK_SEC  0xffff0000
   #define MASK_FRAC 0x0000ffff


/*********************************************
  QoS specific values
*********************************************/

   //video related
   #define TC_V1 25
   #define TC_V1S "96 kb/s"
   #define TQV1 96
   #define SLEEPV1 1

   #define TC_V2 24
   #define TC_V2S "48 kb/s"
  #define TQV2 48
  #define SLEEPV2 2

  #define TC_V3 23
  #define TC_V3S "32 kb/s"
  #define TQV3 32
  #define SLEEPV3 3

  #define TC_V4 22
  #define TC_V4S "24 kb/s"
  #define TQV4 24
  #define SLEEPV4 4

  #define TC_V5 21
  #define TC_V5S "19.2 kb/s"
  #define TQV5 19
  #define SLEEPV5 5

  //audio related
  #define TC_A1 12
  #define TC_A1S "96 kb/s"
  #define TQA1 96

  #define TC_A2 11
  #define TC_A2S  "5 kb/s"
  #define TQA2 5
  #define SLEEPA2 19

////////END////////////////////////////

// Global functions to control
// LookUpTable's mutex operations.

void getLookUpMutex();
void releaseLookUpMutex();


#endif
```