# AN ADAPTIVE FRAMEWORK FOR END-TO-END QUALITY OF SERVICE MANAGEMENT

LIFENG ZHOU
*(B.S. and M. Eng., Nanjing University)*

A THESIS SUBMITTED

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

NATIONAL UNIVERSITY OF SINGAPORE

2008

# Acknowledgement

First and foremost, I wish to express my deepest gratitude to my supervisor, Dr. Pung Hung Keng and co-supervisor Dr. Ngoh Lek Heng for their invaluable guidance and support throughout my research efforts towards this thesis. Their insights and suggestions to the problems in this thesis have enlightened me in various detailed aspects throughout the work.

Dr. Ooi Wei Tsang, Dr. Samarjit Chakraborty and Associate Professor Roger Zimmermann have served as my reviewers at different stages of this thesis. I would like to express my appreciation for their suggestions and comments and their time in reviewing this thesis.

I would like to thank all my colleagues in the Network Systems & Services (NSS) Laboratory. Among them, special thanks go to Dr. He Jun, Dr. Gu Tao, Dr. Long Fei, Dr. Chen Lei, Ms. Feng Yuan, Ms. An Liming and Mr. Suthon Sae-Whong for their constant assistance and encouragements.

Last but not least, I would like to thank my parents and my wife for their love, unconditional support and patience during the course of my doctoral studies.

# Table of Contents

# List of Tables

# List of Figures

# Summary

High-speed networks and powerful end-hosts enable new types of Quality of Service (QoS) sensitive applications such as Video-On-Demand to be offered. In contrast to traditional text and data applications which are burst and elastic in nature, these emerging real-time multimedia applications are demanding on system resources such as bandwidth and CPU, and are also sensitive to continuous QoS performance. To provide end-to-end QoS to users, researchers have spent great efforts in finding suitable QoS provisioning mechanisms in areas such as QoS middleware, adaptive applications and QoS-aware networks. We find that the approaches of most existing researches have been piecemeal, wherein each focusing on a different aspect of the QoS provisioning mechanisms. We argue that the real design issue of end-to-end QoS is more complex than when each of these QoS mechanisms is considered on its own. It is therefore not sufficient to rely merely on, say middleware, applications or networks to fulfill end-to-end QoS. Instead, an integrated approach to the overall end-to-end QoS provisioning, harmonizing QoS mechanisms in the applications, middleware and networks are essential.

In this thesis, we propose an adaptive end-to-end *QoS coordination and management framework* (QCMF) for the QoS management of multimedia applications. Unlike other end-to-end QoS architectures which mainly focus on the interface design between adjacent layers, resource reservation or work-flow management, QCMF aims at designing an effective end-to-end QoS platform for accommodating and coordinating QoS efforts from heterogeneous end-to-end QoS components (e.g., end-host QoS management and network QoS provision). Our solution encompasses existing or new QoS mechanisms at three levels: the network level, the middleware level and the application level, each of which is abstracted as a meta-model in the end-to-end QoS scenario where their behaviors and interactions are studied. The proposed framework is adaptive in the sense that it recognizes and coordinates the adaptive behaviors of multimedia applications and networks in view of the changing runtime environment context. Besides, QCMF provides the ability of dynamic composition of end-hosts' communication stacks, which provides another possible dimension of QoS adaptation at the middleware level.

With the aforementioned methodology in mind, we have proposed a set of techniques to fulfill our overall design objectives of a coordinated end-to-end QoS management. Firstly, we propose a unified knowledge plane for end-to-end QoS modeling, in which QoS information of each end-to-end QoS component is described semantically. The semantic approach of modeling QoS knowledge facilitates the deployment of multimedia applications in heterogeneous environments where services of desirable (or compatible) features can be selected according to runtime service availability. Moreover, information sharing among QoS components becomes easier as different end-to-end QoS components would have a common understanding of QoS knowledge while interacting with each other. Secondly, we propose a novel approach to the analysis of QoS violations. By monitoring end-to-end flow statistics and application performance, a QoS violation can be quickly identified with high accuracy. Such an approach outperforms traditional rule-based violation detection methods which have seldom undergone a rigorous testing procedure and require clear margins of QoS parameters in asserting a QoS violation. Lastly, we propose an end-to-end QoS coordination scheme and algorithms for runtime collaborative end-to-end QoS management. By exchanging QoS information and coordinating adaptation behaviors among QoS components, a QoS violation can be solved by either a local adjustment at the QoS component where the violation takes place or being processed by another QoS component participating in the end-to-end collaboration. Such a decision is made at end-hosts in a pure end-to-end fashion without violating the end-to-end design principle of the Internet. Our prototype implementation validates our design philosophy and demonstrates that QCMF is functional. Performance evaluation results of the prototype show that QCMF works effectively in many aspects of end-to-end QoS management such as control signaling, knowledge processing, violation detection and coordinated adaptation.

# Publications

[1] Lifeng Zhou, Lei Chen, Hung Keng Pung and Lek Heng Ngoh, “Identify QoS Violations through Statistical End-to-end Analysis”, to *submit to a journal*, 2009

[2] Lei Chen, Lifeng Zhou and Hung Keng Pung, “Universal Approximation and QoS Violation Applications of Extreme Learning Machine”, *accepted by Neural Processing Letters*, Springer, 2008

[3] Lei Chen, Lifeng Zhou and Hung Keng Pung, “Universal Approximation Analysis and Applications of Orthonormal Neural Networks”, *submitted to International Journal of Pattern Recognition and Artificial Intelligence*, 2008

[4] Lifeng Zhou, Lei Chen, Hung Keng Pung and Lek Heng Ngoh, "End-to-end Diagnosis of QoS Violations using Neural Networks", *in Proc. the 33rd IEEE International Conference on Local Computer Networks (LCN)*, 2008

[5] Lifeng Zhou, Hung Keng Pung and Lek Heng Ngoh, “An end-to-end Framework for Coordinated QoS Adaptation”, *in Proc. the 33rd IEEE International Conference on Local Computer Networks (LCN)*, 2008

[6] Lei Chen, Lifeng Zhou and Hung Keng Pung, “Approximation Capability of Feedforward Neural Networks with Least Square Solutions”, *accept with major revision by NeuroComputing*, 2008

[7] Lifeng Zhou, Hung Keng Pung, Lek Heng Ngoh and Tao Gu, “Ontology Modeling of a Dynamic Protocol Stack”, *in Proc. the 31st IEEE International Conference on Local Computer Networks (LCN)*, 2006

[8] Lifeng Zhou, Hung Keng Pung and Lek Heng Ngoh, “Towards Semantic Modeling for QoS Specification”, *in Proc. the 31st IEEE International Conference on Local Computer Networks (LCN)*, 2006

[9] Lifeng Zhou, Hung Keng Pung and Lek Heng Ngoh, “Knowledge Modeling for End-to-End QoS Management”, *in Proc. the 1st International Conference on Communications and Networking in China (ChinaCom)*, 2006

[10] Liming An, Hung Keng Pung and Lifeng Zhou, “Design and Implementation of a Dynamic Protocol Framework”, *Computer Communications,* Volume 29, Issue 9, pp. 1309-1315, May 2006

[11] Suthon Sae-whong, Hung Keng Pung and Lifeng Zhou, “QMan: an Adaptive End-to-End QoS Architecture”, *in Proc. the 12th IEEE International Conference on Networks (ICON)*, 2004

# CHAPTER 1 INTRODUCTION

Most conventional and legacy network applications have been designed to operate in low to moderate speed (a few 10s of Mbps) network environments. These networks have been useful and adequate for supporting text and data applications, including distributed applications requiring short requests-responses, in which relatively small amount of bandwidth is needed for each transmission. Moreover, these applications are rather elastic in nature, i.e., they can tolerate great variations in performance such as packet delay and throughput rates. In recent years, great advance has been made in communication technologies where networks that can support data traffic in gigabits per second on every port (e.g., Gigabit Ethernet) are now available off-the-shelf. As a result, high-speed networks and powerful end-hosts enable new types of applications such as Video-on-Demand, multimedia-based collaborative computing and teleconferencing. In contrast to traditional elastic data applications, these emerging multimedia applications have different traffic characteristics and are demanding on system resources such as network bandwidth and CPU time slice. The challenges in designing such applications generally lie in catering for time dependent (or continuous) media: audio and video. Besides storage speed, memory size and processing power, timely delivery of media data over networks is also an essential factor. It requires not only considerable computing resources, but also ensures that these resources will be available over a certain period of time. Failure to sustain such provisioning will generally compromise the presentation quality of continuous media. Thus, the need for transmission quality assurances for these *Quality of Service* (QoS) sensitive applications arises naturally.

The question of which suitable mechanisms for the provisioning of QoS has been asked and possible answers suggested and issues debated. One interesting but trivial suggestion of solution is through over-provisioning of bandwidth. The motivation of this thought is that bandwidth, due to increasing availability of fibers and wavelength-division multiplexing (WDM) technique, is potentially abundant and cheap. We believe over-provisioning can greatly ease QoS problems but is not a panacea. This is because of at least three main reasons: (1) not all QoS problems are constrained by bandwidth, jitter is a classical example; (2) no matter how much bandwidth the network can provide, new innovative applications is likely to be created in the near future to consume them [1]; (3) unless there is a common physical transmission technology (fiber is a potential candidate) for all different network solutions, the vision of the abundant bandwidth cannot be materialized for a very simple reason: all networks are to be interconnected in one way or the other and hence those networks of lower bandwidth will become the QoS bottleneck. Indeed LANs, dial-ups, wireless LANs, WANs and broadband co-exist and interconnect to form the global Internet. The use of high-speed core networks has not eliminated QoS problems, as we have known and experienced today. For example, wireless communication technologies, including wireless LAN (IEEE802.11b/g), Bluetooth and 3G mobile networks, are being developed and deployed as common services nowadays, which enables wireless multimedia streaming to be delivered in light-weighted devices such as handset. As end-applications are very likely to run over either fixed wired networks or wireless networks, the overall network environment becomes more dynamic and heterogeneous. Hence the QoS problems are more difficult to be resolved by relying on a simple mechanism, such as over-provisioning of bandwidth. Suitable QoS mechanisms are needed in networks and end-hosts to best assure the timely delivery of multimedia data.

## 1.1 MOTIVATION

For over ten years, researchers have proposed various QoS solutions in either end-hosts or networks. In QoS provisioning through networks, researches have been focused on providing suitable QoS models and service disciplines, as well as appropriate admission control and resource reservation protocols. For example, the *Internet Engineering Task Force* (IETF) has defined several standard QoS architectures such as *Integrated Services* (IntServ) [2], *Differentiated Services* (DiffServ) [3], *Constraint-based Routing* [4] and *Multiprotocol Label Switching* (MPLS) [5]. IntServ and DiffServ are well-known network QoS models, which have been studied and compared (through simulation, prototyping and performance measurements) by many researchers. IntServ, relying on the *Resource Reservation Protocol* (RSVP) [6], duels with resource allocations and reservations for each data flow and hence would have the potential to provide guaranteed QoS service. Many network vendors, such as Cisco and Sun Microsystems have IntServ/RSVP implementations on their routers [7]. On the other hand, DiffServ is based on a simple model where traffic entering a network is aggregated into classes and treated differently within a DiffServ-enabled network. There are router prototypes and products actually implementing DiffServ service. MPLS is a forwarding scheme that has the ability to aggregate traffic flows and hence can provide a basis for both IntServ and DiffServ QoS support over core networks. Constraint-based routing intends to address QoS from the routing point of view by establishing an appropriate route meeting some QoS constraints such as bandwidth or/and delay requirements.

In end-hosts, various QoS architectures have been proposed and discussed. According to their resource management styles, these solutions can be categorized into *reservation-based approaches* and *adaptation-based approaches* [8]. Reservation-based approaches employ resource reservation and admission control mechanisms (such as CPU preemption and scheduling) to guarantee the availability of resources before multimedia data is delivered [9]. The sustaining of transmission quality depends on the QoS technologies of the underlying platform (e.g., QoS capability of the operating system and network), in which the data packets are handled. Nevertheless, because of the following reasons, multimedia transmission cannot rely solely on such resource allocation and reservation mechanisms.

- QoS degradation in best effort networks is often unavoidable [10], as QoS assurance provided by the underlying services may vary from time to time.
- The Internet traffic produced by end-users exhibits a dynamic behavior. There has been no effective QoS reservation mechanism for dueling with the diverse QoS requirements of applications and the dynamic behavior of the network traffic.
- QoS guaranteed technologies have yet to be established as common services, hence most today's networks are still operating in best-effort or best assured mode.

In view of the above restrictions, QoS adaptation, which allows a multimedia application to react suitably to occurring QoS violations, is essential to ensure that the application can sustain certain level of QoS in various runtime environments. An adaptation-based QoS approach can operate in best-effort or QoS-enabled network environments and manages QoS in a pure end-to-end fashion where QoS monitoring, analysis and adaptation are enforced throughout the lifecycle of the transmission to smooth the quality fluctuation and best maintain the agreed QoS level. An adaptation-based approach requires minimum modification to existing network architecture, thus makes itself more suitable to be deployed over current non-real-time OS and best-effort network environments (or future QoS-enabled network environments).

In adaptation-based QoS researches, progresses have been made in several directions such as *QoS-aware applications*, *QoS middleware* or *QoS-enabled operating systems*. Most work done in the application layer is related to the transmission of continuous media streams (e.g. variable bit rate codec, media compression, frame-dropping and layered encoding scheme), and hence is rather media specific and restrictive in certain application domains [11][12][13][14][15]. On the other hand, researchers have also proposed research prototypes of QoS enabled/sensitive operating systems, applying results from real-time scheduling theory to support system level QoS management [16][17][18]. However, such an approach would often result in a proprietary OS, which is therefore not popular. In recognition of these limitations, more active research efforts have been devoted to provide QoS supports as middleware services [19][20][21][22][23][24]. The QoS middleware approach is popular for at least two main reasons despite of its performance overhead: (1) the QoS solutions are likely to be independent of the network and OS platforms, and (2) the QoS controls can be specifically designed and possibly be transparent to applications.

This thesis proposes an adaptive *end-to-end QoS Coordination and Management Framework* (which we call QCMF) for QoS management of end-to-end multimedia transmission. Different from most existing work that focuses on a particular QoS provisioning domain (e.g., networks or applications), QCMF provides an integrated solution for end-to-end QoS management which designs a set of techniques to embrace existing or new QoS efforts from different areas of end-to-end provisioning. Details of our approach will be discussed later.

## 1.2 Problem Statement

The need to provide QoS support for networked multimedia applications has long been recognized and discussed. As QoS issue has not been part of the design considerations of virtually all network architectures, including that of the Internet, the design and development of suitable QoS provisioning mechanisms has to be carefully considered so as to ensure the stability of current Internet architecture and its compatibility with other add-on network services such as Network Address Translation (NAT) [27]. In fact, the complexity of QoS provisioning has already resulted in various QoS solutions each focusing on a different aspect of the QoS provisioning mechanisms, depending on the perspectives and design centric of the designers. As discussed, these solutions can be broadly classified into three main design viewpoints: QoS-aware applications, dedicated QoS middleware and network QoS models.

However, the real design issues of QoS provision are far more complex than when each of these design viewpoints is considered on its own. This is simply because meeting performance requirements of QoS-sensitive applications is fundamentally an end-to-end issue. It requires all QoS-enabled facilities along the end-to-end path working cohesively to achieve the desired end-to-end performance. As most existing QoS solutions focus on their respective areas while paying little attention to the interaction with other QoS services on the end-to-end path, QoS can only be sustained in their local domains, while no satisfactory end-to-end performance can be provided to end-users. In this sense, we believe that a more holistic approach to the overall end-to-end QoS provisioning, integrating QoS mechanisms in the applications, middleware and the networks is essential.

An adaptive QoS coordination and management framework (QCMF) has been developed based on such a design consideration. The framework embraces QoS services along the provisioning path and provides mechanisms for QoS coordination and adaptation among them in both build-time instantiation and runtime QoS management. Different from existing integrated end-to-end architectures which have typically developed a whole set of new end-to-end QoS mechanisms by themselves [9][28], QCMF aims at accommodating existing QoS techniques from different domains and providing a platform for their interaction. For instance, QCMF does not invent any new signaling protocol for QoS negotiation among end-to-end QoS components (opposite to [29]), but makes use of any existing protocols capable of negotiation. Unlike [30] which designs its own network QoS implementation as part of its end-to-end QoS efforts, QCMF assumes a generic network service differentiation model for end-to-end collaboration. Such a model can be easily mapped to existing standard network QoS models such as DiffServ which is built on the same basic QoS discipline of service differentiation. In this way, QCMF requires minimum modification of current network architecture and hence has a better chance to be accepted and implemented as common utility services over the Internet.

## 1.3 THESIS CONTRIBUTIONS

This thesis proposes an adaptive QCMF framework for QoS management in end-to-end multimedia transmission. The solution embraces existing and new QoS mechanisms at three entity levels: networks, middleware and applications. QCMF provides necessary management functions that include, for example, QoS negotiation, monitoring and adaptation. With runtime adaptation, QCMF enables multimedia applications to best maintain certain degree of QoS under constrained system and network resource availability. In summary, this thesis makes the following key contributions:

1. We propose a new design philosophy with respect to how current communication architectures of end-hosts and networks could be modified to accommodate end-to-end QoS services. Rather than designing a new set of QoS mechanisms for each communication layer so that they can be seamlessly integrated together for end-to-end QoS provisioning, we propose to unify existing isolated QoS solutions at different layers so as to fulfill end-to-end QoS requirement. We believe our solution is easier to be implemented and deployed in current network environment.

2. (As the continuation of point 1) we propose a set of techniques to enable the collaboration among end-to-end QoS sub-systems. We treat each of the QoS sub-systems as a meta-component and design an end-to-end framework and methods for accommodating and supporting interactions and dynamic adaptations among them. In this context, we are not participating in the performance enhancement of QoS mechanisms of any individual layer. Instead, our contribution is to provide a platform for harmonizing and coordinating existing QoS mechanisms in applications, middleware and networks in the context of overall end-to-end QoS provisioning.

3. We propose a uniform semantic approach and meta-models to abstract QoS characteristics of applications, middleware and networks. Each of these meta-models will provide consistent interfaces so as to facilitate interactions among adjacent QoS models. Based on such a semantic specification method, we establish

a knowledge plane for QoS information exchange among different QoS-subsystems for the benefit of end-to-end QoS negotiation and management. The advantages of such an approach lie in a powerful and expressive method for specification as well as an easy way for information processing, matching and sharing.

4. We propose a novel end-to-end approach to QoS management with respect to the diagnosis of QoS violations. By monitoring end-to-end flow statistics and application performance, a QoS violation can be quickly identified with high accuracy as we have tested. Such an approach outperforms a traditional rule-based violation detection method which has seldom undergone a rigorous testing procedure and requires clear threshold values of QoS parameters in asserting a QoS violation.

5. We demonstrate the design concepts of points 1-4 and the functionality of the proposed QCMF framework through prototype implementation. We have developed a set of software reflection techniques for the implementation of meta-QoS models for applications, middleware and networks. In addition, decision-making algorithms, heuristics and policies have been defined for a collaborative end-to-end QoS management. Through physical measurements of our implementation, we have shown that QCMF can achieve the aforementioned features and functionalities with acceptable overhead.

## 1.4 Thesis Outline

The rest of the thesis is organized as follows.

Chapter 2 surveys relevant literatures in areas of end-host QoS research and network QoS research. We discuss the features and limitations of existing approaches. The differences between QCMF and previous work are also compared.

Chapter 3 gives an overall picture of QCMF by explaining its design philosophy (i.e., design reference model), system architecture and management functionalities.

Chapter 4 elaborates the knowledge modeling in QCMF whereby characteristics of each QoS sub-system with respect to end-to-end collaboration are semantically abstracted and processed.

Chapter 5 explains our approach for runtime QoS monitoring and violation analysis. We also give an overview of the violation identification algorithms we have engaged, whose performances are compared and discussed in Chapter 7.

Chapter 6 presents the cross-component adaptation scheme in QCMF. Detailed description about our design assumptions, meta-models for end-to-end QoS components and coordination algorithms and heuristics are explored. Simulation models and results are then introduced which has validated the correctness of our approach.

Chapter 7 describes our prototype implementation and performance measurements of QCMF.

Chapter 8 concludes the thesis and discusses future work.

# CHAPTER 2
# LITERATURE REVIEW

The open problem of QoS provisioning has been addressed by various research efforts in the past years. In this chapter, we will review some of the advance in both network and end-host QoS researches. More comprehensive end-to-end QoS solutions such as cross-layer architectures and integrated end-to-end QoS systems will also be introduced and compared. By examining these related researches, we will show the advantages of our work over previous studies.

## 2.1 QOS IN COMMUNICATION SYSTEMS

The term QoS is first introduced to describe characteristics of low-level data transmission in communication systems. With the appearance of distributed multimedia applications, the meaning of QoS has been re-defined as "the collective effect of service performance which determines the degree of satisfaction of a user of the service" [31]. In general, QoS represents a set of quantitative and qualitative characteristics of a distributed multimedia system that are necessary to achieve the required functionality and performance of an application. Here functionality and performance refers to both the proper delivery of media data to a multimedia application user and the overall user satisfaction [32].

In practice, QoS is often expressed using measurable *QoS parameters*. A QoS parameter describes a specific attribute of a communication system or a performance requirement of a multimedia application. Each QoS parameter can be viewed as a typed variable with bounded values. An application's QoS requirements are conveyed in terms of high-level QoS parameters that specify what the application desires. These QoS parameters are assessed by the underlying communication system to determine whether application requirements can be met or not.

The underlying system needs resources to promote its service to multimedia applications. Essentially, there are two kinds of resources relevant to the performance of a multimedia application: *end-host resources* and *network resources*. The former consists of processing power, memory, data buffer in an end-host and its peripheral multimedia devices; the latter includes network bandwidth and packet queuing priority. To manage these resources for applications, two camps of QoS researches have been established focusing on their respective areas, namely end-host QoS research and network QoS research.

## 2.2 QoS Provisioning Architectures

The open problem of providing end-to-end QoS support has been addressed by various research efforts in the past years [9][28][32][33]. This section reviews existing QoS researches applicable to areas such as network QoS, end-host QoS and end-to-end QoS.

### 2.2.1 Network QoS Models

To support QoS in the Internet, IETF has defined several standard service models and mechanisms to meet the demand for QoS. The IntServ/RSVP [2][6] architecture intends to provide end-to-end bandwidth reservation by maintaining per-flow state information along the path from the flow sender to the receiver. However, the complexity of per flow operations usually increases as a function of the number of flows. In addition, it is difficult to maintain the consistency of per flow state in a distributed network environment. Thus the IntServ model is not scalable to large networks [1]. Such a scalability problem has resulted in the DiffServ approach [3] where QoS is achieved by a coarse level of service differentiation among a small number of traffic classes. The main advantage of DiffServ over IntServ is that core network will only operate on aggregated flows instead of per flow in IntServ. In edge routers, packets are processed and aggregated on the basis of service classes. However, the DiffServ solution will become complex when QoS is to be offered over multiple DiffServ domains. Notably, there is a widely used QoS reference model merging these technologies. This includes the models combining IntServ in access network and DiffServ in the backbone network [34]. MPLS [5], on the other hand, is a layer two forwarding scheme that has the ability to aggregate traffic flows and hence can provide a basis for both IntServ and DiffServ QoS support over the core network.

Network QoS research in recent years mostly focuses on (1) the functional improvement of these standard QoS models through techniques such as traffic engineering [35][36], or (2) discusses the impact of these models on existing communication facilities such as the performance variation of TCP protocol [37]. Nevertheless, we should note that network QoS models or solutions discussed above can only deliver end-point to end-point QoS, i.e., from the network egress point of a flow sender to the ingress point of a flow receiver. However, the main body of QoS communication lies within both end-hosts and their applications. In another word, what we want to satisfy is the QoS requirements from multimedia applications, which is more precisely, application-to-application QoS. The network QoS models by themselves, can not provide application-to-application QoS. A simple example is that, the fluent delivery of video frames to an end-user relies on network resources such as bandwidth and end-host resources such as CPU time slice. While a network QoS may assure the provision of bandwidth, the successful end-to-end QoS provision still depends on the sufficient CPU time slice allocated at both flow sender (for media encoding) and receiver (for decoding). The gap between application-to-application QoS and network QoS is left for end-host QoS to bridge. Moreover, QoS is not always fully guaranteed in these proposed network QoS models. For instance, DiffServ provides a sense of resource allocation and service differentiation, but it never guarantees the provision of QoS in the network: intra service class bandwidth contention in a DiffServ domain is often managed by traffic engineering technologies such as statistical admission control [38] and *Random Early Detection* (RED) [39]. It is obvious that such traffic engineering technologies cannot strictly guarantee even network-wide QoS. Thus an end-to-end flow may expect temporary quality fluctuation during transmission where end-host QoS mechanisms may take their places.

### 2.2.2 QoS-aware Operating Systems

A number of pioneering efforts have produced useful QoS provisioning mechanisms in end-hosts, among which QoS-aware operating system research has once been a focus. To support the execution of real-time multimedia applications, the operating system of a computer has been argued to have the ability to manage and resolve resource contentions of these applications so as to ensure timely processing and delivery of multimedia data.

Several research prototype operating systems have emerged, applying results from real-time scheduling theory. For example, the DASH kernel [16] uses an admission control algorithm based on a timeline and then uses earliest deadline scheduling to actually sort all tasks. In order to guarantee the performance of an application, computational requirements of the application need to be measured beforehand and be analyzed together with its timing constraints such as delay bounds. In this way, an application can be executed within expectation where its timing constraints can be satisfied. Similar observation can be found in RT-Mach [17] and Pegasus [18] where applications need to specify timing constraints explicitly. Based on that information, the OS kernel can calculate its CPU usage and provide fine-grained timestamp and synchronization.

In recent years, great strides have been made to support QoS provisioning in commercial OS and network products. Most Windows operating systems are now able to signal RSVP and do kernel level packet scheduling [40]. There are also several add-ons available to win32 platforms which can provide advanced QoS supports such as CPU resource reservation [41]. On the other hand, large network vendors, such as Cisco and Sun Microsystems have embedded DiffServ on their high end routers [42]. However, as the Internet today is still best effort, there is no means to reserve network resources such as bandwidth, which is vital to end-to-end QoS provisioning. Thus these low level (OS and network) QoS supports are still tentative and premature in nature.

### 2.2.3 QoS Middleware

Traditionally, middleware is a layer of software that runs above heterogeneous operating systems and communications systems, providing a uniform interface to distributed applications. In end-host QoS researches, various projects have been proposed to provide QoS supports as middleware services. Typically, a QoS middleware provides services ranging from QoS specification, negotiation to runtime supervision. The following paragraphs will provide a detailed discussion on some of the latest QoS middleware and compare their key features with those of our QCMF.

DaCaPo++ [43] is a middleware QoS project that supports a range of multimedia applications. It automatically configures itself at start-up time to provide suitable communication protocols and multimedia oriented services that are adaptable to application needs. MCF [20] from the same research group offers flexible multipoint communication services through protocol configurations at start-up time. To make QoS parameters more application friendly, “types” of media can be specified in both MCF and DaCaPo++ where different treatment will be provided to each media type. On the other hand, DJINN [24] and Chameleon [44][45] are designed to support runtime protocol stack re-composition in addition to build-time composition, which offers more flexibility of QoS adaptation than DaCaPo++. DJINN allows application developers to create and connect model components in the form of connection diagrams. At runtime, such a component graph can be modified if intra-components reconfiguration can not solve a QoS violation. In a heavy loaded network environment, for example, the congestion control mechanism of TCP may introduce unnecessary overhead to a multimedia stream which can tolerate certain degree of packet loss. Through runtime re-composition, a TCP protocol component can be replaced with light-weighted protocol such as UDP in DJINN. Leveraging on the *dynamic protocol framework* (DPF) [46] component, our QCMF provides similar build-time stack composition and runtime re-composition compared with DJINN. In the context of QCMF, DPF offers a possible dimension of QoS adaptation at the middleware level. However, the QoS adaptation issue (e.g., end-to-end information sharing and decision-making) in QCMF is more carefully designed compared with aforementioned researches in that it also reviews adaptation choices in other domains such as multimedia applications (e.g., variable video frame rate) and networks (e.g., service class upgrade in DiffServ). In this sense, QCMF offers a more comprehensive end-to-end solution where middleware level adaptation is only one of the runtime considerations.

The $2K^Q$ project [47] from UIUC proposes a resource-aware service configuration model for heterogeneous distributed environments. $2K^Q$ employs multi-tie QoS translation. Firstly, specification of the application is translated into a set of component configurations. Secondly, the set of component configurations are translated into QoS-aware component specification (QoSCSpec). Lastly, QoSCSpec is translated into the corresponding system QoS parameters and their resource requirements (e.g., CPU or network bandwidth). The QoS specification and mapping process of QCMF is similar to that of $2K^Q$. However, QCMF proposes a systematic semantic model to describe the roles and relationships among various QoS entities including middleware components, network QoS services and application requirements. As a result, standard high level QoS entities can be more easily matched and mapped into system level resource specification.

Agilos [22] is a middleware control architecture to assist application-aware adaptations. The main contribution of this project is to introduce a fuzzy control model for the decision-making of QoS adaptations. The correctness and efficiency of their model have been proven by mathematical analysis and prototyping. Agilos utilizes fuzzy rules in the form of "IF-THEN-ELSE" clause to define adaptation behavior. However, as system complexity increases, reliable fuzzy rules and membership functions used to describe system behavior are difficult to determine. Comparatively, QCMF engages a machine learning approach to QoS violation analysis. By examining the end-to-end flow statistics and application behavior, QCMF can identify a QoS violation without the need to specify threshold values for communication parameters. Moreover, Agilos is specifically designed for those applications that receive control commands from the middleware. Hence, Agilos does not allow applications to specify fuzzy rules as adaptation decision is solely made by analytical translation through middleware probe service [8]. A similar approach is taken in [48] which defines strategic and tactical QoS managers. Strategic QoS managers take a global view of QoS provided by a set of application components within the manager's policy domain while tactical QoS managers provide local control over application components. In contrast to these studies, there is virtually no restriction on the kind of multimedia applications that QCMF can serve. For those applications that have their own QoS logics which are out of the control of a QoS middleware, QCMF provide information support by establishing a knowledge plane for information record and exchange (Chapter 4). For other applications that do not have built-in intelligence for QoS management, QCMF will guide the behavior of these applications through end-to-end coordination. In both scenarios, QCMF allows application-specific policies to be defined, which is used to direct the management behavior and adaptation decision-making of the end-to-end QoS system (Chapter 6).

Through reviewing these recent researches, we have identified the following trends in the design of emerging QoS middleware. Firstly, QoS middleware are becoming more and more flexible. Many QoS middleware today are designed in component-based architectures, meaning that various functionalities are encapsulated into components and can be swapped in and out on the fly [24][49][50]. In this way, higher flexibility can be achieved where customized services can be provided to a multimedia application. Secondly, several QoS middleware has incorporated additional features such as multipoint and security support [23], which makes them more versatile in supporting a wide range of application needs. Lastly, more and more network applications incorporate multimedia contents and require corresponding QoS supports. As a result, purposeful middleware has been proposed to serve a specific application in a particular environment [51][52]. For example, [53] has designed a distributed middleware for networked audio and visual home appliances, which is executed on commodity software. Built on Linux platform, such a middleware can control a wide range of home appliances.

### 2.2.4 Multimedia Applications and Media Framework

As stated earlier, most QoS researches in the application layer are related to the transmission of continuous media streams, and hence are rather media specific and restrictive in certain application domains [11][12][13][14][15]. A multimedia application typically supports various codecs for media compression such as Motion JPEG, MPEG-4 and H.264. These codecs present diverse visual-auditory quality to an end-user by incorporating different compression techniques and compression ratio. On the other hand, different codecs have different emphasis on resource allocation. Theoretically, a highly compressive codec requires more CPU time slice for media compression and less network bandwidth for data transmission compared with a low compression ratio codec. Hence multimedia applications can choose codecs of different resource requirements so as to fit into runtime environments of diverse conditions and resource availability.

A multimedia application in networking environments generally will present delay-sensitive and loss-tolerant characteristics [54]. Firstly, most multimedia applications can cope with certain amount of packet loss depending on the sequence characteristics and error concealment strategies (e.g. packet loss up to 5% or more can be tolerated at times [55]). Secondly, multimedia applications have stringent delay constraints. For interactive applications such as videoconferencing, delay upper bound is commonly known as less than 200 milliseconds. Comparatively, multimedia streaming applications can tolerate delay up to 1 or 5 seconds [56]. Typically, data packets that arrive after their display time are discarded at the receiver side or, at best, can be used for concealing subsequently received multimedia packets. The delay-sensitive, resource-intense and loss-tolerant features of multimedia applications suggest that QoS management and adaptation can be effective in adjustment of a multimedia application's presentation quality in view of runtime dynamics.

To assist the design and deployment of multimedia applications, media framework has been proposed to provide a semantically rich programming environment and facilitate the access of I/O device and synchronization of different media streams. *Windows Media Technology* (WMT) [57] and *Java Media Framework* (JMF) [58] are two popular media frameworks. Platform independent and open source are the advantages of JMF over WMT. JMF enables audio, video and other time-based media to be added to Java-based applications and can capture, play, stream and transcode multiple media formats. It also supports RTP/RTSP [59][60] in order to interoperate with standard-based, third-party video streaming servers from, for example, Apple, Sun and Kasenna. Hence, our prototype implementation of QCMF has chosen JMF as the development platform.

### 2.2.5 Cross-layer QoS Architectures

Layering is a common approach for dealing with the high complexity of QoS provisioning, so that research issues of each layer can be considered in isolation. Existing QoS literatures mainly deal with QoS provisioning within the context of one of the individual architecture layers as aforementioned. A QoS researcher in this way would typically focus on one aspect of the QoS provisioning mechanisms for a layer, neglecting possible related QoS mechanisms in others. For example, current end-host QoS solutions tend to adapt their middleware or applications to the changing network QoS conditions. Thus an ongoing session may have to be aborted when the resource scarcity in network (e.g., bandwidth shrink) degrades the initially agreed QoS to a level beyond any end-host adaptation can cope with. However, we argue that network in this case may simply be a better place to exercise QoS adaptation (if the network is QoS-enabled such as a DiffServ network) so as to prevent the abortion of the session. This example clearly illustrates a serious shortcoming of dealing QoS problems in isolation, which leads to a less effective end-to-end QoS solution. Hence, we assert that any decent end-to-end QoS solution must consider the interactions of QoS mechanisms between layers.

A number of cross-layer QoS architectures have been proposed to address the QoS issue by assuming a centralized solution with a single management point and direction of decision-making[61][62][63]. A cross-layer framework jointly analyzes and optimizes the different strategies available at various system layers (e.g., physical layer, medium access control (MAC) layer, network/transport layers or applications). For instance, authors of [64] employ a central coordinator to decide QoS configurations in three layers of an end-host (i.e., application task, OS scheduler and CPU speed). It should be noticed that the management scope of most cross-layer proposals are within one end-host where fine-grained control of different layers can be achieved. Although such a federal solution works for local decision-making within one end-host, it may not be applicable to end-to-end QoS provisioning in that a local coordinator in one QoS subsystem is unlikely able to decide QoS configurations and adaptations for others (such as the network or a remote host). In view of this, QCMF tries to coordinate QoS efforts from various sub-systems for the benefit of end-to-end provisioning rather than determining their respective configuration and actions.

### 2.2.6 End-to-end QoS Schemes

As isolated QoS provision may lead to localized QoS solutions which are undesirable for end-to-end QoS delivery, an overall QoS framework that encompasses QoS mechanisms of communication components and facilitates implementation that would harmonize their interactions is ideal for end-to-end QoS transmission. Among the few reported work in the area of integrated end-to-end QoS schemes [65][66][67][68][69], focuses have been put on connecting respective QoS-flows of each architecture layer (e.g., interface design, service negotiation protocols [29], specification and translation) and supporting the underlying enabling mechanisms in each layer. For example, the Enthrone project [30] proposes an integrated management solution which covers an entire audio-visual service distribution chain, including content generation and protection, distribution across networks and reception at user terminals. Similarly, [70] proposes a general QoS management framework to select and configure most appropriate system components according to user requirements and runtime available resources. In [71], authors propose a content-aware bandwidth broker (CABB) to manage QoS for multimedia applications in a DiffServ environment. CABB allocates network resources to multimedia flows based on client requirements, the adaptability of the application, and its tolerance to network level parameters such as bandwidth, delay, and latency. Kim et al. describes an end-to-end performance simulation model and methodology for the CDMA 2000 network in [56]. The simulator models all protocol layers from physical to the application layers. Details of the packet handling characteristics of each network element along the end-to-end path are also considered to compare and measure performance of applications under different settings. However, all these work has overlooked the complexity of end-to-end QoS with respect to decision-making, especially in the case of QoS adaptation.

End-to-end QoS in our view is distributed and heterogeneous in nature; each of its QoS components may have its own QoS mechanisms and adaptation strategies. In this context, for example, QoS middleware may have its own means of adaptation in case of QoS violations. Meanwhile, adaptive applications may also be able to transform themselves to cope with runtime changes. Things will become more complex if the network: (1) is also QoS-enabled where diverse service options are of choices, (2) offers heterogeneous QoS in different network domains, some of which, for example may employ QoS routing while others may make use of load control or selective packet dropping techniques [39]. Given multiple QoS objectives and QoS service options on the end-to-end path, a good (coordinated) QoS decision-making will certainly become more difficult due to an expanded solution space and possible interactions between QoS options. Such a complexity is often not considered in the aforementioned end-to-end schemes. With such a consideration, QCMF is designed to be an adaptive end-to-end framework with emphasis on system-wide coordinated adaptation, leveraging on the capabilities of each end-to-end sub-systems.

## 2.3 Dynamic Protocol Composition

Dynamic protocol framework (DPF) [46] is a middleware component in QCMF, which can provide dynamic protocol stack composition at call-setup time and re-composition (i.e., protocol inserting or swapping) at runtime. DPF provides the flexibility of building a protocol graph of dynamically loaded components supporting media flows, in a manner similar to other component-based frameworks. In the context of end-to-end provisioning, DPF offers one possible dimension of QoS adaptation within the communication protocol stack which can supplement current prevailing QoS solutions at application or network level.

In DPF, protocol components need not to be bound at design time, which provides the flexibility in composition of protocol stacks. Instead of specifying the name of a protocol, applications now can specify their desired QoS properties. For example, a multimedia application may request to reserve resources before its session starts. At build-time, available protocol services that match this requirement (e.g., RSVP or other signaling protocol with similar resource reservation capability) will be selected. Such an approach eases the deployment of multimedia applications in heterogeneous network environments in that if a target protocol is not available, other protocols of similar functions can be selected so that the end-to-end delivery will not fail (e.g., [72]). The flexible composition of protocol stacks also facilitates the QoS adaptation process. For instance, two video codecs may present similar presentation quality to an end-user, but at different compression rate and hence each demands for different amount of network bandwidth. In the case of QCMF/DPF where codec names need not be specified by multimedia applications (instead, media quality such as medium or high should be specified), a codec that requires more bandwidth may be replaced at runtime with another one that consumes less bandwidth in case of network congestion.

To ensure a consistent description of all end-to-end QoS entities, we have designed a semantic scheme for modeling and processing of protocol stacks, which is presented in Section 4.3. The semantic model of communication protocols and protocol instances are also illustrated in Appendix B. The integrity of protocol and protocol stack configuration are ensured with sets of dependencies and supported media formats primitives to be defined by protocol developers. Service dependencies and media format compatibility are checked at stages of both build-time configuration and runtime reconfiguration to ensure the correctness of a protocol stack.

## 2.4 SUMMARY

The complex QoS problem has led researchers to focus on different aspects of QoS provisioning in a fashion similar to the layered approach in network systems design. This has resulted in many rigid QoS solutions each addressing one or very few aspects of the problems with respect to a set of application scenarios, middleware, or networks. These silos of solutions are either too difficult to integrate, or if doable, often would lead to overall inefficiency due to poor coordination between respective QoS sub-systems. Hence, we believe that any satisfactory end-to-end QoS solution must consider the coordination of QoS mechanisms between QoS sub-systems (such as those in end-hosts and networks) and manage them in a cohesive and coordinated fashion. Through comparison and discussion, we have found that most existing end-to-end QoS schemes focus primarily on the configuration issues such as interface design and QoS-flow management, which is essential, but not sufficient for meeting performance requirements of multimedia applications. Motivated by these observations, we propose our ideas of end-to-end QoS collaboration by accommodating and coordinating exiting QoS architectures in applications, middleware and networks. Details of our approach will be discussed in the following chapters.

*CHAPTER* 3

# THE QOS COORDINATION AND MANAGEMENT FRAMEWORK

This chapter gives a high level overview of the architecture and management functions of our QoS coordination and management framework (QCMF). We first present a reference model for end-to-end QoS provisioning and discuss our design philosophy and relevant QoS concepts. Subsequently, we introduce the system architecture and management functionalities of QCMF, whereby detailed description of our research will be presented in the next few chapters.

## 3.1 REFERENCE MODEL FOR QOS MANAGEMENT

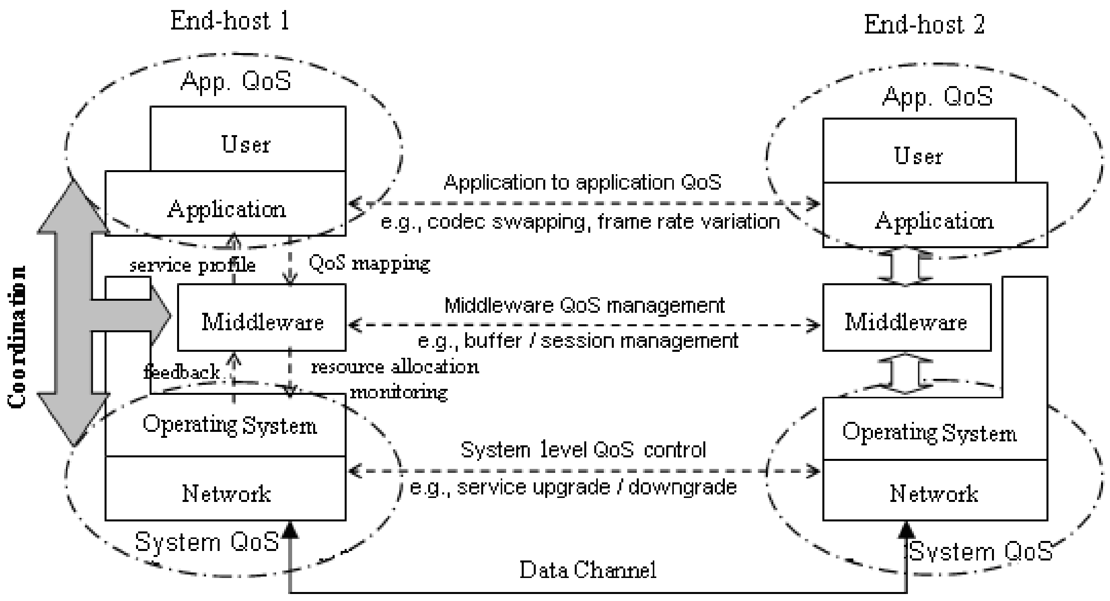


Figure 3-1: Reference model for end-to-end QoS provisioning and coordination

To deal with the complexity of end-to-end QoS provisioning, we introduce a reference model to guide the design of our end-to-end QCMF framework as shown in Figure 3-1. This model outlines relevant concepts and procedures for end-to-end QoS provisioning which can be analyzed from both *architecture* and *management* dimensions.

From the architecture perspective, the model will yield the identification of several abstracted QoS layers and their corresponding roles in end-to-end QoS delivery:

- *System QoS* includes efforts from a host's OS and the network which provides basic data transmission support between end-hosts. Native packet level QoS support can be offered if the OS and the underlying network are QoS-enabled. In addition to data link or MAC level QoS provisioning [73], research concerns in this area have also included network communication level load balancing and fairness issues [74][75].
- *Middleware QoS* offers a rich set of services for the configuration and management of the transmission quality (e.g., buffer management, flow synchronization and QoS-based handover) outside the kernel space of an end-host [21][22][23][24]. Middleware QoS solutions are likely to be independent of the network and OS platforms and hence are able to work over heterogeneous network environments.
- *Application QoS* refers to the ability of multimedia applications to self-configure and respond to the changes of runtime operating conditions or user requirements. As discussed, such abilities are commonly related to the transmission and performance tuning of particular continuous media streams such as audio and video (e.g. variable bit rate codec or layered encoded audio

and video) [14][15]. Hence QoS solutions at application level are rather media specific and restrictive to a certain application domain.

For over a decade, researchers have proposed various QoS solutions which according to their places of research interest, can be summarized into one of the above categories. As has been explained in Chapter 2, QoS researchers in this way would typically focus on their own domains of QoS provisioning while neglect (possible related) QoS mechanisms in others. Such a layered QoS research leads to an independent and local optimized implementation, but would often result in sub-optimal end-to-end performance. In this sense, an overall QoS framework that encompasses QoS mechanisms of various layers is essential for the satisfaction of an end-user. Furthermore, end-to-end QoS in our view would be distributed and heterogeneous in that each QoS layer (subsystem) may have its own provisioning mechanisms and adaptation strategies. An end-to-end QoS framework thus should take into consideration the characteristics and restrictions of each end-to-end QoS sub-system (e.g., QoS layer) so that a sound overall adaptation solution can be identified among multiple available end-to-end choices at runtime.

Based on the above design philosophy, we have arrived at the design of QCMF as an adaptive end-to-end QoS coordination and management framework. Our solution embraces existing and new QoS mechanisms at three entity levels: the network level, the middleware level and the application level. We treat each of these QoS sub-systems as a *QoS component* in our end-to-end framework and try to devise an effective platform and methods for accommodating and supporting interactions and dynamic adaptations among them. In this context, we are not participating in the performance tuning or enhancement in QoS mechanisms of a particular QoS component. Instead, our focus is to provide a management platform for harmonizing and coordinating QoS mechanisms in applications, middleware and networks in the context of end-to-end QoS provisioning. The proposed framework is adaptive in the sense that it recognizes and coordinates the adaptive behaviors of multimedia applications and networks in view of the changing runtime environment context. Furthermore, QCMF provides the ability of dynamic (re-)composition of end-hosts' communication stacks during runtime as an additional way to duel with QoS violations at the middleware level.

From the management perspective, end-to-end QoS delivery and coordination is fulfilled through three inter-related facilities: *QoS knowledge plane*, *QoS control plane* and *QoS data plane*. The data plane is the carrier over which media contents are forwarded; the behavior of the data plane is regulated by the control plane via management functions such as configuration, signaling and adaptation; the enforcement of these control actions relies on the information originated from the knowledge plane which abstracts QoS characteristics of each end-to-end QoS component and infers QoS status of end-to-end QoS transmission. More detailed description about the data plane and control plane can be found in Section 3.2.

To achieve the aforementioned design objectives of QCMF, we first create abstract QoS models for the management of different QoS components (Chapter 4). We shall then understand their interactions and establish the coordination requirements among these QoS models. Subsequently a QCMF framework can be established for implementation of QoS states monitoring, signaling and feedback (Chapter 5), together with algorithms and heuristics for distributed decision-making (Chapter 6).

## 3.2 QCMF MANAGEMENT ARCHITECTURE

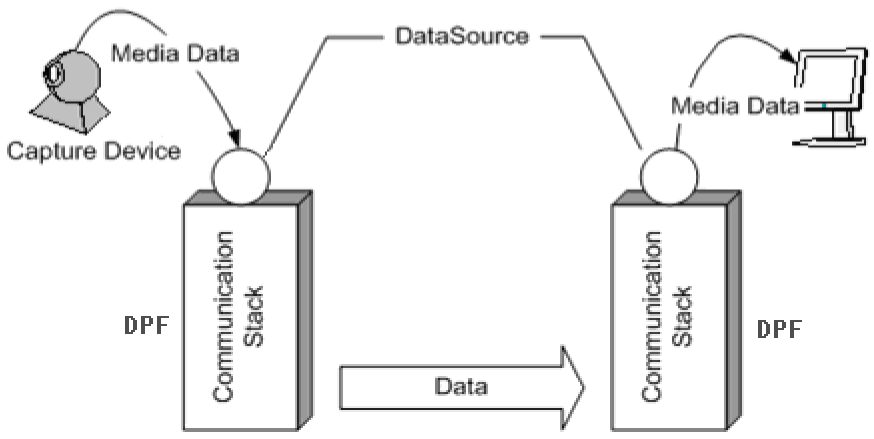


Figure 3-2: End-to-end QoS transmission scenario

A typical scenario of end-to-end QoS transmission is illustrated in Figure 3-2 where media data is delivered from the source host to the destination via the in-between network. The media source can be an audio/video flow either retrieved from a media file from the sender's hard disk or be lively captured from a web camera. The media sink typically is a device that consumes and displays the received media flow. In QCMF, both media source and sink are abstracted as DataSource[1] objects within which media processing functions such as data rendering, encoding, packetization, and decoding are defined and encapsulated. The actual packet delivery is fulfilled in an adaptive data plane which is composed of adaptive communication stacks in two participating end-hosts and the network in between. The communication protocol stacks of the media sender and receiver are adaptive in the sense that they support start-up time semantic composition and runtime re-composition of the stacks (e.g., swapping media codecs) so that media flows of different qualities can be delivered at the cost of either high or low resource consumption (e.g., bandwidth or CPU cycle utilization). Such an adaptation is performed by the dynamic protocol framework (DPF)

[1] DataSource, a term of JMF standard, is used to manage the transfer of media contents. A DataSource encapsulates both the location of media and the protocol and software used to encapsulate the media.

QoS module [46] under the guidance of QCMF whereby a decision is made based on the runtime monitored QoS conditions. On the other hand, the communication network between end-hosts may also be QoS-enabled (e.g., composed of a few DiffServ subnets) whereby different service options are of end-hosts' choices. As an end-to-end QoS management architecture, QCMF embraces all these QoS components as part of the end-to-end provisioning by modeling their QoS abilities for the consideration of build-time QoS orchestration and runtime system-wide adaptation (Figure 3-3).

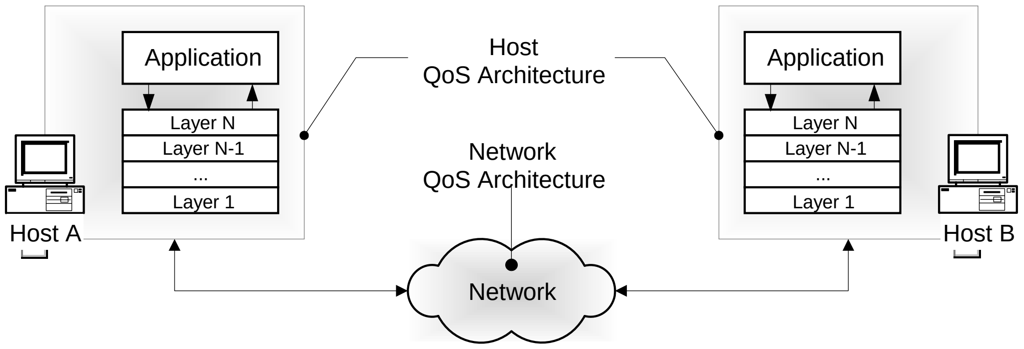


Figure 3-3: QCMF incorporates both host architectures and network architectures

While the data plane handles the actual media data transmission and processing, the QoS control plane, on the other hand, is responsible for the exchange of control information and signaling messages between QoS components (Figure 3-4). Similar to the design in other network protocols and systems, QCMF separates the control plane from the data plane[2]. This is to ensure that control overhead will not degrade the transmission quality of media data. At both media sender and receiver, QCMF establishes a *QoS management middleware* (QMan) respectively. At the media sender, QMan monitors the outputting media flow and cooperates with the receiver's QMan to derive end-to-end QoS status such as end-to-end packet delay and jitter. At the media receiver, QMan analyzes end-to-end flow QoS information and makes adaptation

[2] In our prototype implementation, the control plane is implemented using both Java Remote Method Invocation (RMI) method and native TCP sockets that are separated from data plane sockets (UDP).

decisions against runtime QoS violations. Theoretically, such an adaptation choice could be a media codec swapping in end-hosts, a service upgrade in QoS-enabled networks, a combination of the two or other adaptation choices available in a particular end-to-end environment (e.g., reduce application QoS). The final choice is made through a system-wide evaluation process performed by QMan middleware in a pure end-to-end manner, with details to be presented in Chapter 6. Such an action or decision-making is augmented by the meta-data contained in the knowledge plane, which will be introduced in Chapter 4.

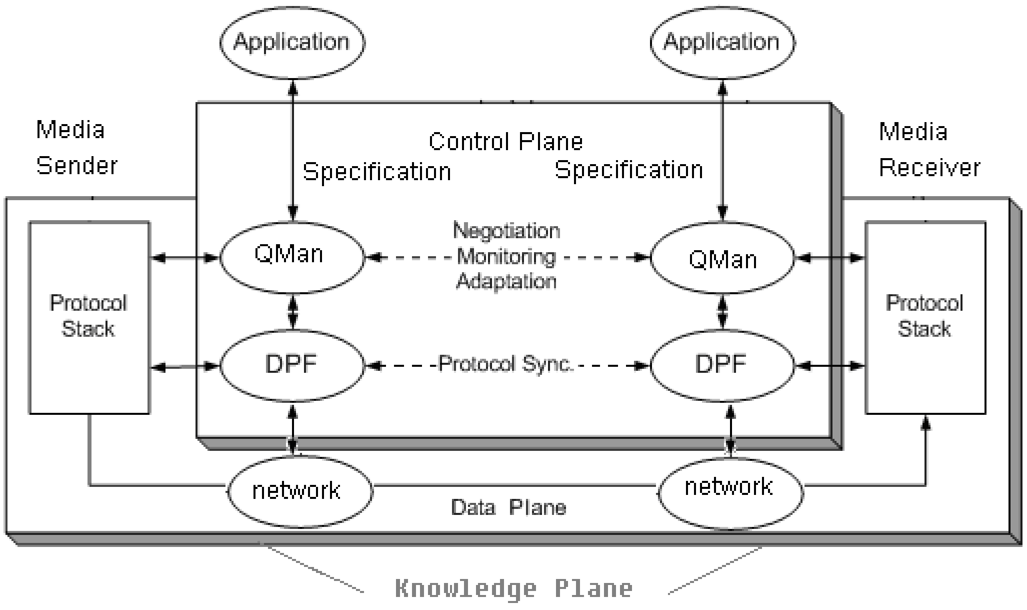


Figure 3-4: QCMF design concepts: control plane for signaling, data plane for media transmission and knowledge plane for meta-data recording

## 3.3 QCMF MANAGEMENT FUNCTIONS

As an adaptive end-to-end QoS management framework, QCMF provides necessary management functions for multimedia applications in a fashion similar to other component-based end-to-end QoS architectures. Such management services include QoS specification, negotiation, enforcement and runtime adaptation. In addition to that, QCMF dips into several end-to-end QoS issues which results in the following research merits that distinguish QCMF from existing end-to-end solutions:

- **Build-time QoS specification.** QoS specification is the process to capture application requirements which are subsequently translated into configurations of the underlying communication system. QCMF adopts a language-based approach for QoS specification whereby complex specification structures and language formats can be defined. Moreover, QCMF defines a semantic model to describe QoS requirements of multimedia applications whereby common knowledge of the same application domain can be formulated and re-used. Such a semantic specification process is handled by the semantic QoS specification (SQS) scheme of QCMF and is to be elaborated in Chapter 4. Finally, the same semantic approach has been applied to the modeling of QoS knowledge of other end-to-end QoS components of QCMF (e.g., networks), which provides a uniform knowledge sharing interface that facilitates their interactions.
- **Adaptation rules and policies.** As a comprehensive end-to-end QoS framework, QCMF acknowledges various adaptation methods from different end-to-end QoS components, some of which may achieve similar end-to-end effect. For example, an end-to-end delay violation may be alleviated by either media buffering technique at the media sink or solved by service upgrade in a DiffServ network environment. In view of this, QCMF allows applications to define policies regarding adaptation preferences and sequences which can guide the runtime selection of adaptation choices and the way application QoS is gracefully degraded. Furthermore, application specific management policies can be specified whereby, for instance, certain application QoS parameters (e.g., video frame resolution) can be traded off first to sustain the quality of other

more pertinent parameters (Chapter 4). As a result, application QoS can be managed in a desirable manner.

- **Information sharing with applications.** In QCMF, applications can query end-to-end transmission status (e.g., end-to-end delay) and QCMF management functions status at any time. In addition, interfaces are defined where QoS status reports can be delivered to the application in an event-driven fashion [76]. Such a design is specifically for those multimedia applications that have built-in adaptation engines and require runtime environment information update from the underlying platform (Chapter 4). For other applications that require external control for QoS management, QCMF will take into consideration their adaptation abilities in making an end-to-end QoS decision-making and coordination (Chapter 6).
- **Runtime QoS management.** Runtime QoS management in QCMF includes QoS negotiation, monitoring, decision-making and adaptation. QoS negotiation is intended for the structural composition of end-hosts' protocol stacks, service agreements with network QoS components (if any), as well as communication parameter configuration (e.g., streaming port number). Runtime monitoring and violation detection is fulfilled through observation and analysis of traffic characteristics of an end-to-end flow (Chapter 5). Based on the analysis results, decision-making is performed within end-hosts with respect to end-to-end coordination and adaptation (Chapter 6). All these runtime QoS management tasks are executed in an end-to-end fashion by QCMF (Figure 3-5) so as to avoid the possible scalability problem in network QoS management.

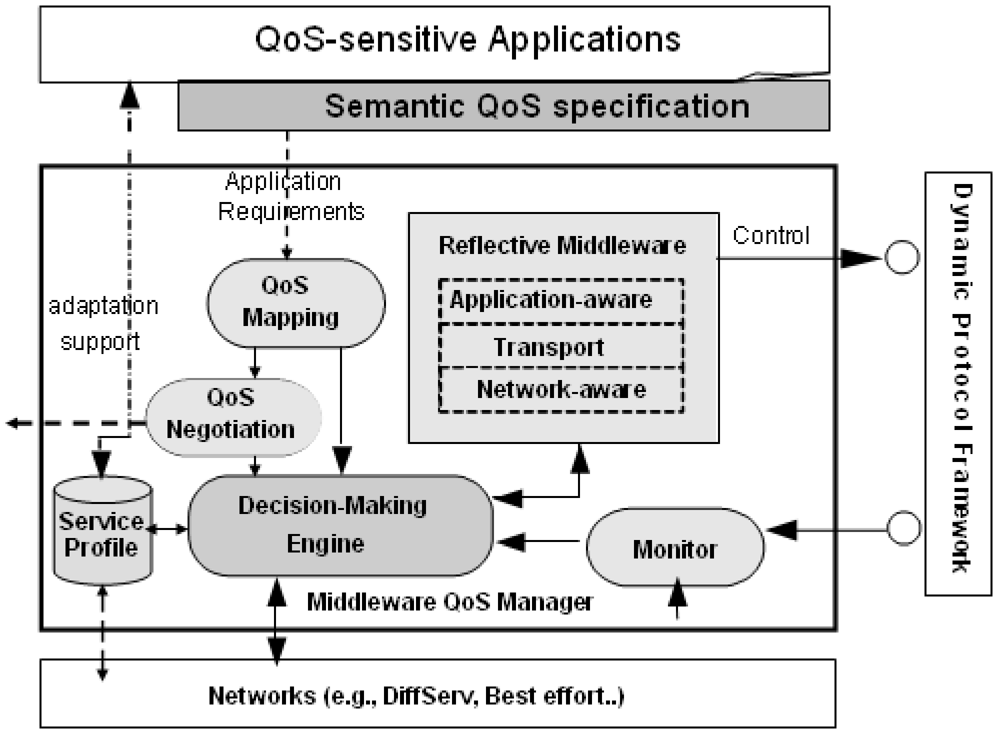


Figure 3-5: Management functions of QCMF are fulfilled by its several build-time and runtime execution modules: Semantic QoS Specification (SQS) for knowledge modeling, Middleware QoS Manager (QMan) for runtime management and Dynamic Protocol Framework (DPF) for middleware level adaptation

## 3.4 SUMMARY

As has been discussed, end-to-end QoS provisioning is not likely to be a simple issue of one communication layer but requires joint efforts from different QoS components along the end-to-end path. With such a design philosophy in mind, we have arrived at designing QCMF as an adaptive end-to-end QoS architecture that supports multiple QoS component models and services at various system levels (i.e., application, middleware and network) and manage them in a cohesive and co-operative fashion. An overview of management architectures and functions of QCMF is introduced in this chapter whereby particular research focuses are outlined and will be elaborated in the following chapters.

# CHAPTER 4
# END-TO-END QOS KNOWLEDGE MODELING

To enable a coordinated end-to-end QoS management, we first model QoS knowledge of each end-to-end QoS component so as to facilitate their information sharing and interaction. Different from traditional API-based approaches that focus mainly on QoS specification for an individual application domain, we propose a uniform semantic-based approach with meta-models to respectively abstract QoS information of all end-to-end QoS mechanisms including those in applications, middleware and networks. The advantages of such an approach include a more powerful and expressive specification method as well as an easy machine processing and end-to-end sharing procedure. In this chapter, we first discuss the motivation of our semantic modeling approach; detailed design considerations and approaches are presented subsequently. Finally, we describe the semantic knowledge meta-models of each end-to-end QoS component.

## 4.1 QOS KNOWLEDGE AND QOS ONTOLOGY

### 4.1.1 Related Work

Traditional data management is achieved by making use of either programming language elements (e.g., read/write of file streams) or special structures such as customized XML. However, these proprietary methods do not consider the semantic meanings and relations of data, and hence is not suitable for managing meta-data featuring a system. Knowledge management of network systems has received much attention in recent years. In [77], Clark et al. has proposed a knowledge plane for automatic discovery and recovery of the Internet architecture, leveraging on AI tools and cognitive systems. Inspired by this idea, several research projects have been established for better network management relying on logical languages such as Prolog [78][79]. These studies provide general guidance for management and maintenance of a specific network infrastructure, which however have not addressed our concern of knowledge classification, modeling and processing for end-to-end QoS provisioning.

In conventional end-to-end QoS provisioning, focuses have been put on the algebraic calculations of QoS parameters and resource scheduling [80], an example of which can be found in [81] that presents a formal model for mathematical calculation of QoS metrics. Our work as proposed in this chapter differs from and supplements these existing approaches by proposing a logic model for QoS provisioning where semantic meanings and relations of various QoS concepts can be compared and reasoned to determine a correct end-to-end QoS configuration. The ontological approach of QoS modeling is initiated from the semantic web community. However, reported work [82][83][84] so far is mostly limited to the description of web services' QoS properties for service matching and selection. In contrast, we focus on modeling a broad range of end-to-end QoS elements for information exchange and distributed QoS coordination, encompassing three architectural levels and covering various stages of QoS provisioning.

### 4.1.2 General QoS Knowledge

Within the context of QCMF, a knowledge plane has been established to describe QoS information of each end-to-end QoS component. For applications, such information mainly refers to application QoS requirements and adaptation strategies. For middleware and networks, QoS knowledge to be recorded includes their QoS capabilities, work-flow information and dynamic runtime QoS status. At build-time and runtime, application QoS requirements will be translated into communication configurations (e.g., codec parameters, UDP port number) and protocol stack compositions in both end-hosts which will be compared and matched by middleware and network service options (to be addressed later).

Conceptually, four categories of QoS knowledge have been identified and defined in the QoS knowledge plane, namely *static knowledge*, *monitored knowledge*, *profiled knowledge* and *deduced knowledge*.

- **Static knowledge** represents system and environment information that is likely to be constant during a QoS session. This includes description of device type, CPU speed, memory size, network service mode, hosts' OS version and so on. The usage of static knowledge is twofold. Firstly, some of the static knowledge defines the maximum computational or presentational capability that a QoS component can offer to its application. For example, the color depth and screen size of a rendering device (e.g. PDA) impose a physical limitation of the display quality for a video stream where software configuration should not exceed. Secondly, the semantic meaning of static knowledge is helpful in the determination of a correct end-to-end QoS configuration. For instance, if a media sink is executed on a resource-scare device through a low-bandwidth link, by default, it may be automatically equipped with a series of light-weight protocols for information presentation and interaction with a media server.

- **Monitored knowledge** refers to those dynamically changing information acquired during runtime. Examples of such information include runtime CPU availability, memory usage, flow throughput, delay, jitter, packet lost rate and CPU usage of the target process. Monitored knowledge in QCMF is updated periodically (or on-demand) and is shared among end-to-end QoS components for tasks such as inferring the occurrence of a QoS violation; an adaptation action will be triggered once a predefined violation pattern matches with runtime observation (Chapter 5), which subsequently gives rise to an end-to-end coordination and adaptation (Chapter 6).
- **Profiled knowledge** as modeled in QCMF at current stage can be summarized into two groups: (1) user and application profile, and (2) protocol profile. User profile stores QoS preferences of end-users for given application sessions. Application profile represents the QoS requirements of applications for an expressive application specification and mapping into the underlying QoS settings. Protocol profile is a kind of software profile characterizing QoS properties (e.g., I/O relations and dependencies) of communication protocols resided in an end-host. Through ontology modeling of a dynamic protocol stack, QCMF can semantically select appropriate protocols for the composition of a protocol stack that meets QoS requirements of an application/user (details to be presented in Section 4.3.3).
- **Deduced knowledge** describes the kind of indirect knowledge that can be derived from aforementioned direct knowledge (i.e., static, monitored and profiled knowledge) through aggregation or reasoning processes. For instance, a

video streaming receiver's runtime QoS status (a deduced knowledge) can be inferred as RED (a status code indicating the need for an urgent attention) if, for example, its video decoding scheme is MPEG-1 (a profiled knowledge) and the current CPU availability (a monitored knowledge) is below a predefined level. Once such an end-to-end QoS knowledge is asserted by QCMF, an adaptation would probably be triggered to solve the violation according to corresponding rules and policies.

Although the actual type and number of QoS knowledge may differ in a real end-to-end system, the general QoS knowledge presented above gives a guideline with respect to the necessary knowledge elements that can be applied so as to achieve the desired end-to-end QoS effect.

### 4.1.3 QoS Ontology and RDFS Schema

QCMF utilizes the abovementioned four categories of QoS knowledge for end-to-end QoS configuration and runtime QoS management. To facilitate information modeling and exchange among QoS components, all knowledge is modeled by *ontology* in *RDF/RDFS* language [85]. The term "ontology" has a long history in philosophy, in which it refers to the subject of existence. In AI literatures, ontology is a formal and explicit description of concepts in a domain of discourse. Ontology provides a vocabulary for representing knowledge about a domain and for describing specific situations in that domain. On the other hand, RDF/RDFS is a W3C standard language to instantiate ontology, which enjoys the advantages of extensibility, modularity, scalability and logic validity. The selection of RDFS instead of another more powerful ontology language - OWL [86] in QCMF is based on a realistic balance between language capability and performance: RDFS is sufficient for modeling small scale QoS knowledge (as can be seen from the next few sections) and is faster than OWL in knowledge processing which is valuable for QoS management in real-time multimedia transmission[3].

Modeling end-to-end QoS knowledge in an ontology-based approach allows us to describe QoS information semantically in a way that is independent of programming languages, underlying operating systems or QoS middleware. The ontologically abstracted QoS knowledge provides a consistent knowledge basis for information sharing and interaction among end-to-end QoS components for the following observations:

- Ontology is a formal description of concepts and relationships, which is expressive in describing objects, their relations and restrictions. Ontology provides a means for formulating semantic meaning and relations of knowledge while other schemas such as XML can only produce a data model (which is a tree).
- The use of ontology enables different QoS components in QCMF to have a common understanding of QoS knowledge while interacting with each other. The identification of the semantic meanings of QoS concepts is essential in a heterogeneous environment where clear differentiation of QoS entities is needed for understanding and interactions among QoS components (e.g., for service negotiation or selection purpose [87]).
- Ontology modeling of QoS enables machine processing and (potential) formal analysis of QoS knowledge through various processing and reasoning mechanisms (e.g., first-order logic, temporal logic). Such a semantic processing of knowledge is a supplement of traditional mathematical calculation of QoS

[3] A migration from RDF/RDFS to OWL is easy to achieve since OWL is backward compatible. OWL has more ontology definitions and primitives which can be employed if necessary.

parameters that focuses on the quantitative attribute of a QoS parameter. Moreover, as an open web standard for easy machine sharing and interpretation, RDFS in our view will have a better chance to be widely accepted and adopted in practice than other proprietary data representation formats.

### 4.1.4 QoS Ontology Predicates

In our QoS ontology, QoS knowledge is represented as standard first-order predicate calculus. The basic primitive has the form of *Predicate (subject, value)*, in which:

- subject $\in$ S*: is a set of names of objects in concern, e.g., throughput, slidingWindow or other QoS entities of interest.
- predicate $\in$ V*: is a set of relations, e.g., hasValue, hasStatus or other property indicators.
- value $\in$ O*: is a set of all values of subjects in S*, e.g., 10, FULL/EMPTY and so on.

Each basic primitive pinpoints an attribute of a QoS object. For example, *hasVaule(throughput, 10)* may record the throughput of a TCP flow while *hasStatus(slidingWindow, FULL)* indicates that current TCP transmission is in full rate. In addition, each primitive can explore the relation of two QoS objects. For example, *isInstance(RedHat, Linux)* indicates that RedHat is member of Linux family; *compatible(Linux, Unix)* specifies that Linux and Unix are compatible operating systems. Finally, pieces of basic primitives can be linked together to form a complex QoS knowledge using boolean algebra (e.g., union, intersection and complement). For example, *hasStatus(accessNetwork, statusVector)* – the summarized QoS status of the access network, can be represented as the integration of *hasvalue(bandwidth, bValue)* $\vee$ *hasRange(jitter, jRange)* $\vee$ *hasRange(delay, dRange)*.

The structures and properties of an end-to-end QoS component are described in an ontology which includes description of classes, properties and their values. The ontology is written in RDFS as a collection of RDF triples, each statement being in the form (*subject, predicate, object*) where subject and object are the ontology's objects or individuals, and predicate as said, is a property relation among them. As a demonstration, a machine-interpretable RDFS description of the access network QoS ontology is shown in Figure 4-1.

```
<rdf:RDF xmlsn:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlsn:rdfs="http://www.w3.org/2000/01/rdf-schema#"
         xmlsn:qcmf="http://lucan.ddns.comp.nus.edu.sg/qcmf#"
>
. . .
<rdfs:Class rdf:ID="accessNetwork">
    <qcmf:linkedTo rdfs:resource="#coreNetwork"/>
    <qcmf:hasMember rdfs:resource="#delay">
    <qcmf:hasMember rdfs:resource="#bandwidth">
    <qcmf:hasStatus rdfs:resource="#statusVector">
    . . .
</rdfs:Class>

<rdfs:Property rdf:ID="hasMember">
    <rdfs:domain rdfs:resource="#accessNetwork"/>
    <rdfs:range rdfs:resource="#bandwidth">
    <rdfs:range rdfs:resource="#delay">
    . . .
</rdfs:Class>

<rdfs:Class rdf:ID="bandwidth">
    <qcmf:hasValue rdfs:resource="...XMLSchema#int"/>
    . . .
</rdfs:Class>
. . .
```

Figure 4-1: Partial QoS ontology for access network written in RDFS

By applying ontology modeling to QoS elements (as described in Section 4.1.1) in each end-to-end QoS component, a knowledge plane can be established within QCMF, which records QoS information of each end-to-end QoS component. The knowledge plane provides necessary knowledge preparation for QoS information interpretation and sharing so as to empower a coordinated end-to-end QoS decision-making. As discussed earlier, the knowledge plane incorporates QoS mechanisms at three generic end-to-end QoS components: applications, middleware and networks whereby their modeling details are presented in the following sections.

## 4.2 APPLICATION QOS KNOWLEDGE MODELING

Application QoS knowledge modeling is to abstract QoS requirements and capabilities (if any) of QoS-sensitive applications. In QCMF, QoS requirements of such applications are presented to the underlying middleware and network for configuration of communication parameters and control of transmission quality; QoS capabilities of applications are exposed to the underlying for runtime adaptation coordination. The understanding of application characteristics is fulfilled build-time by a sub-module of QCMF – SQS, as is explained below. The runtime interaction and adaptation is performed by another QCMF sub-module – QMan middleware, which will be discussed in Chapter 6.

### 4.2.1 Motivation and Design Considerations

The task of application QoS knowledge modeling is commonly achieved through QoS specification by employing either special *application-programming-interfaces* (APIs) or a *language-based QoS specification method* [88]. Most traditional QoS middleware has adopted the static API-based approach [23][51][89] in which QoS requirements of applications are expressed as API parameters. Despite the simplicity of this approach, such a specification methodology is restrictive (as has been shown in various prototypes) in that it is unlikely to design a set of APIs meeting requirements of multimedia applications of different nature. For example, *QoS-aware applications* (which have built-in adaptive mechanisms and may need information sharing and adaptation coordination with the runtime environment) and *QoS-transparent applications* (which purely rely on system level QoS adaptation) possibly need diverse QoS treatments from the underlying. Within the category of QoS-aware applications, multimedia streaming applications and interactive multimedia applications may have varied emphasis on system performance factors such as throughput and delay despite the fact that both of them rely on these resources. In this sense, a systematic QoS specification method enabling detailed specification for applications of different QoS types is desirable. Language-based QoS specification [90][91][92] is a promising approach for meeting such a design challenge. This is because languages are inherently more declarative, expressive and hence potentially can provide detailed specification for applications. For instance, [93] presents an initial insight into QoS specification by specifying QoS as a combination of metrics and policies. In their work, QoS metrics is defined as performance parameters (i.e., timeliness, precision and accuracy), security requirements and their relative importance in a system. However, existing QoS specification languages remain at *syntax* level, i.e., they are strong in the representation formats (e.g., language validation) but weak in the specification processing capabilities such as semantic specification interpretation and comparison. For example, HQML [94] defines a set of generic XML tags (e.g., ServerCluster, LinkList) for specification of multimedia application QoS. However, it lacks a formal model to identify and classify applications of different kinds so as to provide a customized support to each of them. The similar observation can be found in QoS Modeling Language (QML) [95] in which static language structures (i.e., contract, contract type and profile) are defined. These language elements represent abstractions of application QoS of a general meaning, but can not depict application QoS features of a particular context. For instance, the definition of *reliability* contract type for an audio streaming application (counting packet lost ratio) leveraging on TCP transmission is surely different from that of a complex online banking service (measuring transaction successful rate) which employs various reliable modules possibly covering from the application, middleware proxy to the link layer. Such difference arising from diverse application contexts can not be reflected in a generic language abstraction defined in QML since a black-box approach is taken to treat application QoS. Hence we conclude that these languages cannot provide detailed and customized specification support for applications of various QoS categories.

Another key aspect of application QoS knowledge modeling is the application policy specification. Policies allow applications to describe preferred actions to be taken in case that an application QoS requirement is not satisfied in certain circumstances. Such actions may include where to send a notification or tradeoffs among different performance metrics. However, we argue that such application level policies may not be adequate enough to cover all possible runtime variations. This is because current and future Internet computing is getting more and more complex and heterogeneous where combinations of different end-devices (e.g., a light-weighted PDA or a powerful desktop), networks (e.g., wired or wireless) and software (e.g., a standard 5-layer protocol stack or a lightweight protocol stack) will produce runtime platforms of different performance and characteristics. Surely, application level policies which are formed offline at application design stage can only provide partial and coarse-grained guidance for problem-solving in some of the runtime scenarios. All these facts suggest that other sources of QoS knowledge need to be identified. On the other hand, we acknowledge that end-users can also express their preferences in QoS adaptation. However, realizing that end-users may not wish to be bothered by intricate QoS management issues (or even not quite clear about the details), it would be much better if runtime QoS systems can exhibit some kind of intelligence for automatic response to most QoS problems based on known or derived QoS knowledge. Our application QoS specification framework aims to provide such a means for (semi)automatic end-to-end QoS regulation with minimum external intervention.

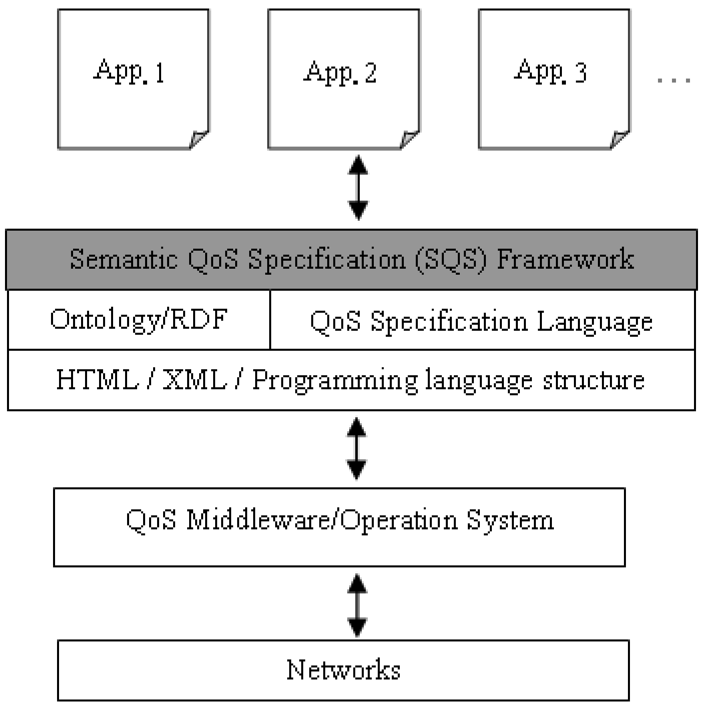

App. 1
App. 2
App. 3
. . .
Semantic QoS Specification (SQS) Framework
Ontology/RDF
QoS Specification Language
HTML / XML / Programming language structure
QoS Middleware/Operation System
Networks

Figure 4-2: Semantic modeling and syntactical QoS specification in QCMF

Based on the above considerations, we propose a *semantic QoS specification* module (SQS) inside QCMF, which emphasizes on (1) application QoS classification and semantic modeling for detailed QoS specification and clear-cut QoS mapping, and (2) domain specific QoS knowledge definition and accumulation for automatic QoS management. It should be noted that SQS is a semantic QoS specification model for a better understanding of application QoS and its interaction with the underlying platform. SQS is positioned on top of existing syntax level QoS specification languages, which provide language structure support for specification and focus on the grammatical correctness and usability of the language elements (Figure 4-2). We have chosen ontology/RDFS to implement our SQS prototype for the aforementioned advantages although in principle any QoS specification language can be employed to instantiate SQS after appropriate extension. The main features of SQS will be introduced in the following sections.

## 4.2.2 Two Layer Application QoS Ontology Model

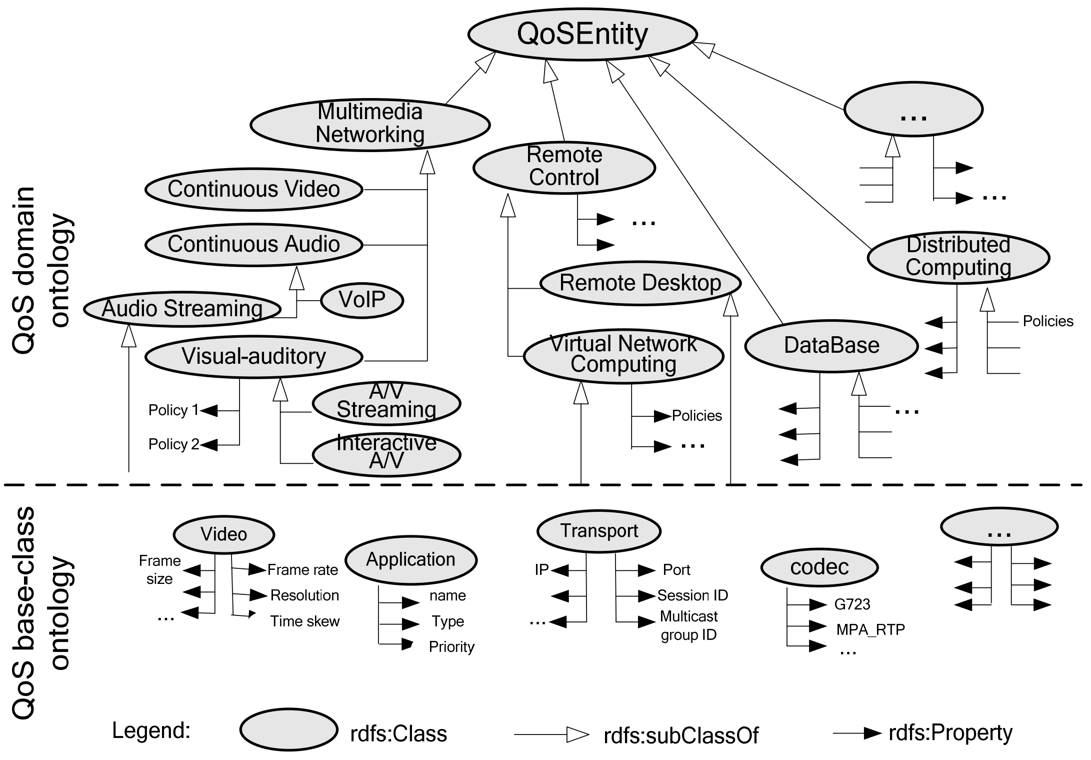


Figure 4-3: The hierarchical application QoS ontology model

To enable a detailed understanding of application QoS and to provide an accurate QoS specification, we propose a hierarchical and extensible application QoS ontology model as shown in Figure 4-3. The model is composed of two layers (divided by the dotted line), each of which is modeled by ontology in RDFS. The lower layer defines semantics of the *QoS base-class ontology* which is the building block for the upper layer *QoS domain ontology*. Each QoS base-class represents an aspect of QoS and comprises relevant QoS parameters that profile QoS characteristics of that dimension. For example, the video base-class defines parameters that describe the presentation quality of a video flow such as frame rate and resolution. The final definition of each QoS base-class is dependent on the QoS middleware and runtime platform. For example, if one QoS middleware can only support audio streaming, then the video related base-classes (including video codecs) will not appear in its QoS ontology model. Hence, it is possible in practice that different QoS middleware defines ontology models of diverse QoS base-classes. However, the semantic meaning of QoS parameters in each base-class should be explicit and consistent.

The upper layer of the QoS ontology model defines application oriented QoS domain ontology. The diverse nature of applications implies the need for a vast set of QoS models. To make application QoS interpretable and manageable, we define an *application-QoS-classification-tree model* in this layer which sorts applications according to their QoS characteristics. Applications with similar QoS characteristics are grouped into the same QoS domain and different QoS domains can be setup for applications of different QoS types. The upper part of Figure 4-3 shows such a classification tree, in which an entry point QoSEntity and four first-level QoS domains are defined (i.e., multimedia networking, remote control, database transaction and distributed computing domains). These four preliminary domains provide a high level abstraction of application QoS which is further classified into sub-domains for a detailed QoS analysis. For example, multimedia networking domain can be classified into continuous audio domain, continuous video domain, and visual-auditory domain, where the last one may be further extended into A/V streaming domain and interactive A/V domain. These two sub-domains differ from each other in their media characteristics (e.g., stored vs. live media), while inheriting the same QoS characteristics (A/V transmission) from their parent domain. By categorizing and modeling, a hierarchical and extensible application QoS classification tree can thus be setup by QoS middleware where new QoS sub-domains are defined as leaf nodes of the tree.

### 4.2.3 QoS Domain Specification and Knowledge Acquisition

Essentially for each QoS domain, we define a *QoS domain specification* that records common QoS requirements of applications in that domain. For instance, an example QoS specification for A/V streaming domain is shown in Figure 4-4, which lists common QoS properties of A/V streaming applications. Each QoS domain specification acts as a library to be build-time linked with application inputs so as to form a complete application QoS requirement statement. The formation of a QoS domain specification is basically realized by merging relevant lower level QoS base-class ontology. Alternatively, domain specification of a child domain can be derived and refined from that of its parent domain. For example, QoS domain specification of continuous-video domain can be established by combining ontology of video base-class, application-general base-class, transportation base-class and codec base-class. Such integration is performed in an offline manner each time a QoS middleware is launched. The derivation of QoS domain ontology from QoS base-classes ontology and the establishment of a QoS classification tree are guided by configuration rules. SQS allows the customization of rules for middleware to produce new QoS domains and (occasionally) modify the definitions of existing QoS domains.

```
<rdf:RDF xmlsn:rdf=”http://www.w3.org/1999/02/22-rdf-
syntax-ns#”
         xmlsn:rdfs=”http://www.w3.org/2000/01/rdf-
schema#”
>

<rdfs:Class rdf:ID="Application-general"/>
<rdfs:Class rdf:ID="Transportation"/>
<rdfs:Class rdf:ID="Codec">
    <description>contains available codecs</description>
</rdfs:Class>
...
<rdfs:Property rdf:ID=”supportedVideoCodec”>
    <rdfs:domain rdf:resource=”#Codec”/>
    <rdfs:range rdf:resource=="...XMLSchema#string"/>
    <rangeValue>G723</rangeValue>
    <rangeValue>JPEG</rangeValue>
    <rangeValue>H263</rangeValue>
…
</rdfs:Property>

<rdfs:Class rdf:ID="Policy1">
    <description>domain knowledge</description>
    <priority>HIGH</priority>
    <rdfs:domain rdf:resource=”#Throughput”/>
</rdfs:Class>
...
```

Figure 4-4: Partial QoS domain specification for video streaming applications

Despite the overhead to construct a hierarchical application QoS classification tree, QoS domain specification can be tailored just to the need of a specific application type. For example, each QoS domain may define domain-specific parameters to reflect the QoS characteristics of applications in that domain. In addition, a QoS domain specification may also contain policies about domain specific QoS knowledge which will have implications on the runtime configuration and adaptation. For instance, the visual-auditory QoS domain may contain knowledge about the significance of audio, video and text components to the perceptual quality of end-users. Figure 4-5 shows an example of such knowledge which states that video components should be subject to flow regulation prior to other media components (e.g., an audio component) in visual-auditory applications. The definition of such knowledge is based on the observation that people are more sensitive to distortions in audio display than in video display in multimedia consumptions [96]. Recognizing such knowledge, QoS middleware runtime should first reduce the quality of the video flow while maintaining the quality of the audio stream, in the case of insufficient bandwidth by, for example, swapping to a less bandwidth-demanding video codec. Unless otherwise specified by end-users, such knowledge can be employed as a default solution to deal with runtime bandwidth shrinking problem.

```
<rdf:RDF
  xmlsn:="http://lucan.comp.nus.edu.sg/SQS#"
  xmlsn:rdf="http://www.w3.org/1999/02/22-rdf-
syntax-ns#"
>

<Policy1 rdf:ID=flow_regulation>
  <priority>deafult</priority>
  <rdfs:domain>video-auditory</rdfs:domain>
  <description>flow priority in A/V streaming</
description>
  <action>swap(video_flow)</action>
</Policy1>
...
```

Figure 4-5: An example of knowledge built in the video-auditory QoS domain

The above QoS knowledge contained in the visual-auditory QoS domain specification will automatically be applied to its child domain - A/V streaming QoS domain. In addition to such knowledge inheritance from parent domain, A/V streaming QoS domain may further define domain particular knowledge for itself, an example of which is presented below in first order logic predicates:

- application (QoS domain, video streaming) $\wedge$ monitor (AudioQuality, ‘LOW’) $\wedge$ userDefinition (AudioPerception, ‘HIGH’) $\vdash$ middleware (Adaptation, throughput)

As has been revealed by some researches, non-interactive auditory applications (e.g., audio flows in an A/V streaming) are more sensitive to throughput than other factors such as end-to-end delay and jitter [54]. In the A/V streaming QoS domain, such a property can be described by the above QoS knowledge, which indicates the sequence of adaptation (e.g., latency to be compromised first) in case of unsatisfactory audio quality.

To conclude, the benefit of detailed partitioning of QoS domains largely lies in the observation that comprehensive QoS specification (i.e., QoS parameters and policies) can be realized in sub-domains to further reflect QoS requirements of applications of a particular kind. As QoS requirements and corresponding resource demands of most today's QoS-sensitive applications have been extensively studied and resulting QoS knowledge been continuously validated, the principle of semantic QoS classification as proposed here is feasible and essential for the systematic modeling of application QoS requirements.

| GPRS QoS class | QoS attributes | Implemented QoS profiles | | | |
|---|---|---|---|---|---|
| | | Streaming | Conversational | Interactive | Background |
| Precedence | Congestion packet discard probability | Tolerable ($<10^{-2}$) | Tolerable ($<10^{-2}$) | Loss sensitive ($<10^{-5}$) | N/A |
| Delay | Latency | Bounded ($<500$ ms) | Bounded ($<80$ ms) | Less stringent than that of conversational & streaming | N/A |
| | Jitter | Stringent | Stringent | N/A | N/A |
| Reliability | Packet loss probability | Tolerable ($<10^{-2}$) | Tolerable ($<10^{-2}$) | Loss sensitive ($<10^{-5}$) | N/A |
| Mean and peak throughputs | Throughput | Guaranteed | N/A | Guaranteed | N/A |
| | Burstiness | Low | High | Higher than conversational | N/A |

Table 4-1: QoS profiles for mobile multimedia applications

The domain specific QoS knowledge can be acquired in QCMF in three ways. Firstly, system administrators are assumed to contribute system-wide management policies for domain regulation (e.g., what kind of traffic will be regulated or prohibited). Secondly, QoS knowledge for classic applications can be established through theoretic and experimental studies from the research and industry community. For example, an in-depth research has been done in [54], investigating into the QoS requirements of advanced applications and their effects on networks. Table 4-1 shows another example of QoS profiles for QoS-sensitive applications in a GPRS environment as is proposed and tested in [97]. Lastly, QoS domain knowledge can be derived from continuous validation through experiments. For instance, recognizing that QoS violations of the similar nature may present similar symptoms at runtime, we employ some AI techniques to identify and associate a new QoS violation pattern with known QoS violations so that the same remedy could be applied to solve the new problem. Through application, feedback and validation process, those effective prescriptions to new QoS violations would be acknowledged by the QoS management system, which could then be turned into new QoS knowledge (Chapter 5).

It should be noted that several recent literatures have also suggested grouping of applications based on their QoS requirements for various purposes. For example, [23] defines audio, video and text flows for multimedia applications and designs APIs accordingly. In [98], applications are classified into three classes - conventional Internet services class, playback streaming class and conversational streaming class - each of which is modeled with four attributes: delay, jitter, data rate and packet lost to match with network QoS parameters. As these taxonomies assume a fixed number of application classes, limited benefit could be derived from such coarse-grained classifications. Comparatively, the number of application domains in SQS is not pre-fixed. The support of new application areas can be achieved by either creation of new application types or inheritance and refinement from existing application domains.

### 4.2.4 QoS Compilation and Mapping

A QoS domain specification serves as a template for instantiating an *application QoS specification* (AQoSpec) for a real application of that QoS domain. The QoS specification process in SQS is achieved in the following steps.

Firstly, SQS will analyze the target application for the purpose of fetching a corresponding QoS domain specification at build-time. The determination of a correct QoS domain basically relies on its QoS type claimed by the application (e.g., visual-auditory QoS type). However, such a syntax-based matching may not always identify the most precise QoS domain for the application since each QoS middleware is likely to define QoS classification trees of different depth (depending on the capability of the QoS middleware and runtime platform as explained before) where more detailed specification support for an application may be possible in some sub-domain. To enable a more precise QoS domain matching, we have designed a light-weighted semantic engine that utilizes machine learning and information retrieval techniques to identify appropriate QoS domains for applications [99].

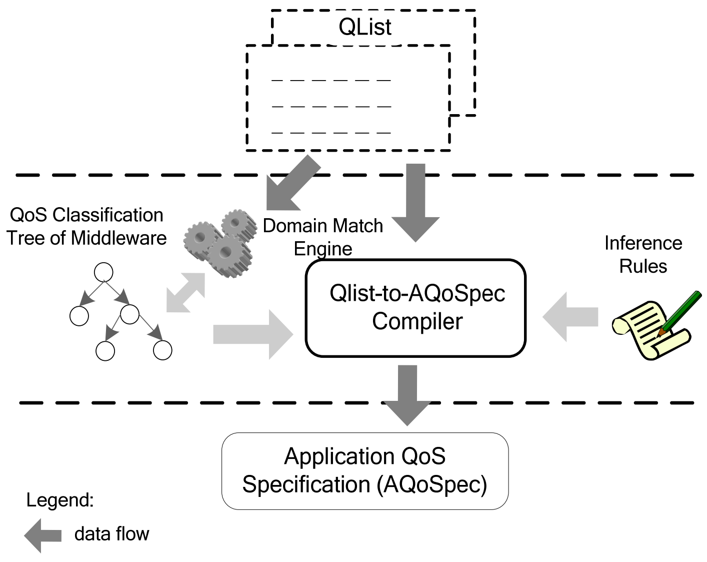

QList
QoS Classification Tree of Middleware
Domain Match Engine
Inference Rules
Qlist-to-AQoSpec Compiler
Application QoS Specification (AQoSpec)
Legend:
data flow

Figure 4-6: Dynamic compilation of AQoSpec

After a most precise QoS domain has been found, SQS will secondly extract application QoS requirements (e.g., values or ranges of parameters) from inputting application QoS documents. We call such a document a *QList*, which produces application QoS requirements including pertinent QoS parameters and associated adaptation policies. Such a QList is preliminary and incomplete in nature because (1) an application designer has not exhausted all QoS configurations of the applications (as discussed earlier, such a complete configuration theoretically is not feasible due to the dynamics of runtime environments), (2) an application designer may not be an expert to figure out all optimal application configurations and QoS degradation choices. In either case, all QoS definitions specified in the QList are compiled according to the corresponding QoS domain specification. Those undetermined parameter values or adaptation strategies will be examined by QoS middleware by incorporating relevant QoS knowledge from corresponding QoS domains[4]. The output of QoS compilation process - an AQoSpec, will provide QoS middleware the most comprehensive information for the runtime configuration and supervision. The whole compilation process is illustrated in Figure 4-6.

An AQoSpec, which is encoded in RDFS for easy machine interpretation, is basically composed of two parts: QoS parameter definitions and policies. Two kinds of QoS parameters are defined in an AQoSpec: *quantitative* and *qualitative* parameters. Quantitative parameters are countable in nature, which at runtime will be translated into resource demands by either analytical or probing QoS mapping mechanisms (much work has been done in this area, e.g., [8]). On the other hand, qualitative parameters mainly refer to the more fuzzy types of QoS requirements such as reliability, security, and availability. At build-time, these parameters will mostly be

[4] End-users runtime may also express preferences which have higher priority than default QoS strategies and application settings. Such a user input is entertained through a separate GUI.

interpreted into middleware and communication protocol stack settings, which will be addressed in the next section. An AQoSpec may also contain QoS domain specific knowledge (expressed as policies) that helps to perform QoS configuration and adaptation at runtime. For example, the knowledge of input/output relations of standard codecs (e.g., MPEG-1, and H.263) is stored in codec QoS base-class, whose resource requirements can be automatically calculated without the need of QoS probing.

## 4.3 MIDDLEWARE QOS KNOWLEDGE MODELING

Middleware QoS knowledge modeling abstracts the capabilities and restrictions of QoS middleware for information sharing and adaptation coordination with other end-to-end QoS components. At current stage of QCMF, our modeling emphasis is on the knowledge description of QMan middleware (e.g., adaptation strategies and capabilities) and the semantic modeling of a host's communication protocols and protocol stacks though it could be extended to include other aspects (e.g., buffer management or session management properties) of a middleware system.

### 4.3.1 Design Considerations

Programming in a typical networked computing environment would have to explicitly specify the communication protocols needed in designing an application. Such a static approach potentially hinders the portability of applications on different runtime platforms: once a protocol stack is established at build-time, runtime re-assembly of the stack to deal with transmission quality variation is normally not supported. In view of this, much work has been done in the area of flexible protocol stacks [24][25][26][49][100], among which our dynamic protocol framework (DPF) is one of the research effort for dynamic protocol stack composition. In the context of QCMF framework, DPF offers one possible dimension of QoS adaptation within the communication protocol stack (e.g., switch from a computational intensive codec to less CPU intensive one in case of CPU overload) which can supplement current prevailing QoS solutions at application or network level. The adaptations of DPF (i.e., runtime re-composition of stacks) are triggered by QCMF as the outcome of a coordinated decision-making process among different end-to-end QoS dimensions and are transparent to end-users. Details about DPF and the differences among all these work with respect to the build-time functional stack construction and dynamic runtime re-composition can be found in [46]. Here we will focus on the semantic modeling of protocol stacks in QCMF by comparing approaches taken in related literatures.

Central to all flexible protocol stack systems is a *protocol knowledge base* (PKB) which is responsible for protocol representation, retrieval and protocol stack composition. PKB keeps the meta-data information (e.g., QoS requirements, capabilities, I/O relations, restrictions and dependencies) of those protocols supported in a system for the benefit of protocol selection and stack validation. However, most work reported so far has taken an ad-hoc approach to design proprietary and exploratory PKB which lacks formality and expressiveness. For example, Rwanda [44] and its successor Chameleon [45] are a series of projects focusing on providing tailored protocol services to media streaming of various types. Rwanda models a protocol stack as a linear list of protocol objects which represent a kind of QoS such as reliable delivery or encrypted communication. All protocol information is implemented by a Java class and is runtime retrieved via reflection for configuration. However, both Rwanda and Chameleon have only designed a few protocol QoS properties for demonstration purpose. The systematic modeling of protocol properties and efficient processing of protocol information for stack configuration has not been considered in their projects.

Dynamic layered protocol stack (DLPS) [101] of Microsoft provides a method to dynamically build a protocol stack for data transfer. A stack description file has been designed comprising a plurality of individual protocol layer description so that the plurality of all these layer description together define which protocol layers will be included in the protocol stack when it is constructed. However, DLPS has not addressed the issue of modeling properties of protocol stacks as a collection of individual protocols. Furthermore, each protocol in DLPS is modeled individually without considering the clustering of protocols of similar natures. For example, all video codec protocols have common properties such as the number of tracks and sampling rate, which can be abstracted as a template for modeling protocols of that category.

In [102], component description is introduced to represent protocol building blocks. Each component is described by a list of provided properties and required properties. The former declares the functionality that can be provided by a component while the latter defines its conditions to be satisfied by others (e.g., downward and upward dependencies). An algorithm to select building blocks is also presented where the solving strategy of stack composition is to match the requested properties of one component with the provided properties of others. However, their work has only focused on the dependency properties of protocols and has neglected the importance of other protocol properties in runtime stack building. For instance, the selection of G.723 protocol or MPEG protocol for audio streaming should depend on the runtime resource availability (i.e., a resource rich environment may choose a codec of better quality which in turn demands for more resources) since dependency is no longer a factor for consideration (i.e., both of the codecs have the same dependency on the underlying protocols in the stack). A selection algorithm considering only one searching factor (e.g., dependency as used in [102]) surely would not find an optimal stack composition in practice.

In brief, PKB design and management functions of all the abovementioned projects have been accomplished by individual efforts making use of either programming languages elements or proprietary data structures. These studies often lack generality and most likely can not provide customized and expressive description for communication protocols and protocol stacks. In line with our semantic modeling of other QoS components in QCMF, we present here our ontology-based protocol knowledge modeling using RDFS to address these shortcomings. Our semantic approach to the modeling and processing of protocol/stack information enjoys the advantages of being expressive, flexible and interoperable with other QoS components in end-to-end QoS knowledge sharing and coordination.

As a middleware service, DPF provides flexible protocol stack composition at session-setup time and dynamic re-composition at runtime. The architecture and workflow of DPF is shown in Figure 4-7. As discussed, the key enablers of DPF include (1) a PKB which stores and represents protocol information ontologically, and (2) a *knowledge reasoner* which holds a reference of protocol meta-data and interacts with other components of DPF throughout the lifecycle of a session for a correct composition and re-composition of protocol stacks. The following sections concentrate on the ontological design of PKB and semantic protocol processing of the knowledge reasoner for the construction of a protocol stack. Particulars about subsequent functional manipulation of protocols (.e.g., protocol insertion or swapping) in DPF can be found in [46].

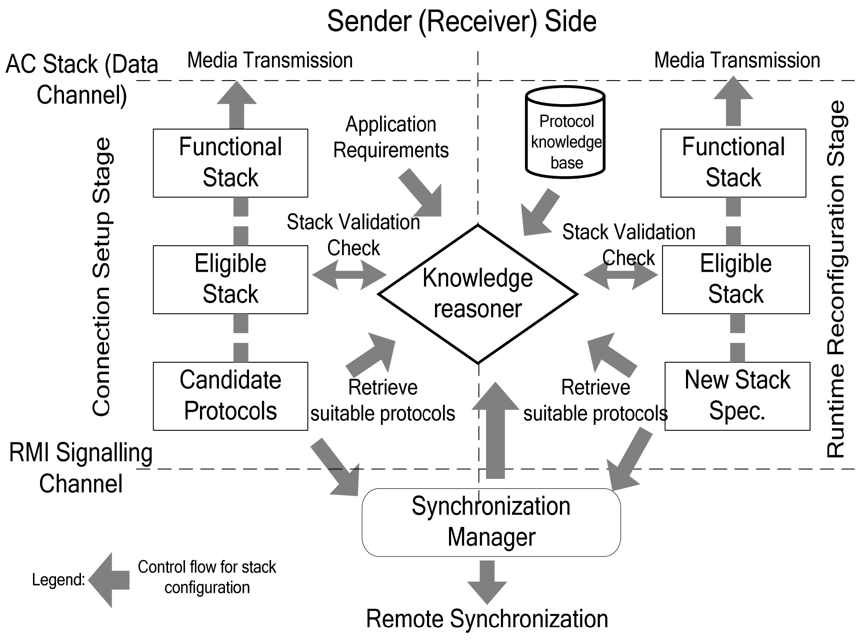


Figure 4-7: Architecture of DPF with ontology modeling

### 4.3.2 Ontology Modeling of Protocols

Since each protocol in DPF provides a specific service, we define an overall *service* class in RDFS expression (Figure 4-8) as an entry point to capture common properties that exist in all protocols. These common properties include layer name (where the protocol lies in the stack), protocol name (what is the protocol), class name (where to find the functional code of the protocol) and so on. Ontology of each protocol will extend this fundamental service class definition.

We then classify protocols into categories according to their functions and positions (i.e., layers) in the protocol stack for an easy characterization and modeling. In each protocol category, common characteristics are abstracted for category-wide re-usage, which is useful in the case of modeling a new protocol of that type. Each protocol category definition extends the fundamental service class ontology and abstracts category specific QoS characteristics. Figure 4-8 shows the ontology definitions of service and category classes. An example property specific to the codec category, which is for media encoding and decoding is also illustrated. The scope that this *supportedInputFormats* property can be applied to is defined by the *rdfs:domain* expression, which provides an isolation of property usage at the grammatical level. The grouping of similar protocols into categories is extensible in that more detailed classification is possible to provide fine-grained ontology description for a small group of protocols of similar nature.

```
<rdf:RDF
  xmlsn:=”http://lucan.comp.nus.edu.sg/SQS#”
  xmlsn:rdf=”http://www.w3.org/1999/02/22-rdf-syntax-ns#”
>

<rdfs:Class rdf:ID="Service"/>
…
<rdfs:Property rdf:ID="className">
   <rdfs:domain rdf:resource="#Service"/>
</rdfs:Property>
<rdfs:Property rdf:ID="protocolName">
   <rdfs:domain rdf:resource="#Service"/>
</rdfs:Property>
…
<rdfs:Class rdf:ID=”Category”/>
<rdfs:Class rdf:IC=”Codec”>
   <rdfs:subClassOf rdf:resource=”#Category”>
</rdfs:Class>
…
<rdfs:Property rdf:ID="supportedInputFormats">
   <rdfs:domain rdf:resource="#Codec"/>
   <rdfs:range rdf:resource="xsd:String"/>
</rdfs:Property>
…
```

Figure 4-8: Protocol knowledge modeling entry point: service and category classes

The ontology of a protocol is built by integration of (templates of) basic service class definition, corresponding category definitions and protocol specific properties. Protocol knowledge to be modeled includes, for example, input/output relations and encoding methods of codec protocols (e.g., VBR, CBR), streaming properties of session protocols (e.g., SIP [103]), reliability features (e.g., availability, security) of control protocols, connectivity characters of datagram protocols and finally, their relations with other protocols (e.g., dependency and compatibility). Figure 4-9 shows a partial ontology definition of TCP protocol for an outgoing session, where characteristics of TCP protocol related to data transmission are recorded. Properties such as *port*, which are specific to transport protocols, are defined within transport category ontology. Other properties such as *slidingWindowSize* are derived from control protocol category (i.e., multiple inheritances are allowed). Meta-data such as *className* and *dependency* are derived from the service class ontology. The protocol ontology for each supported protocol is collectively stored in PKB and handled by the knowledge reasoner to (1) semantically select appropriate protocols that meet (application and environment) QoS requirements, and (2) validate the composition of a protocol stack for build-time construction and runtime re-composition; the details will be presented in the following sections.

```
<rdf:RDF
 xmlsn:=”http://lucan.comp.nus.edu.sg/dpf#”
 xmlsn:rdf=”http://www.w3.org/1999/02/22-rdf-syntax-ns#”
 xml:base="http://www.comp.nus.edu.sg/dpf/Protocol#"
>

<Transport rdf:ID="TCPOut">
   <layerName>protocol.srv.Transport</layerName>
   <protocolName>TCP</protocolName>
   <className>transport.TCPOut</className>
   <upperDepd>NULL</upperDepd>
   <lowerDepdy>Network.IP</lowerDepd>
   <reliability>true</reliability>
   <port>4088</port>
   <polarity>1</polarity>
   <slidingWindowSize>350</slidingWindowSize>
…
</Transport>
```

Figure 4-9: TCP is of (rdf:) type transport and belongs to transport category

### 4.3.3 Semantic Protocol Stack Composition

The overall semantic protocol retrieval and stack composition process is illustrated in Figure 4-10. In end-to-end QoS provisioning, the selection of a target protocol in forming a communication stack is dependent on multiple factors such as the availability of that protocol in an end-host and the preference of the communicating peer. Hence it is meaningless to explicitly bind an application to a specific protocol beforehand. To ensure the runtime portability of QoS-sensitive applications in heterogeneous environments, QCMF allows applications to describe their desired protocol requirements (e.g., resource reservation capability) in their QLists (which has been explained in Section 4.2.4) instead of explicitly designating a specific protocol. Based on the semantics of the application QoS domain and available protocol ontology at runtime, appropriate protocols will be fetched automatically to compose a protocol stack that fulfills application requirements (shown in the left part of Figure 4-10). Note that after the classification and semantic modeling of application QoS by SQS, QoS characteristics of each kind of application is supposed to be well established. The semantic mapping from AQoSpec to middleware and system settings is unambiguous under the same ontology namespace.

In DPF, a protocol stack is a protocol graph that consists of a vector of protocols in the sequence of layers. As a protocol stack is dynamically composed at runtime, stack ontology, which is formed by integrating ontology of member protocols, is an ephemeral model to describe the features of a transient stack. Different combination of protocols will produce stacks with different characteristics. The characterization of a stack is essential for deciding whether the stack composition can suffice application/user requirements or not. For example, <G723, RTP, UDP, IP> is a protocol stack for audio transmission. This stack is characterized by having low perception quality (derived from G723 ontology), unreliable data transmission (derived from UDP ontology) and supports real time session control (derived from RTP ontology). If an end-user can accept medium to low audio perception quality, such a stack can be employed at runtime in case of resource scarcity.

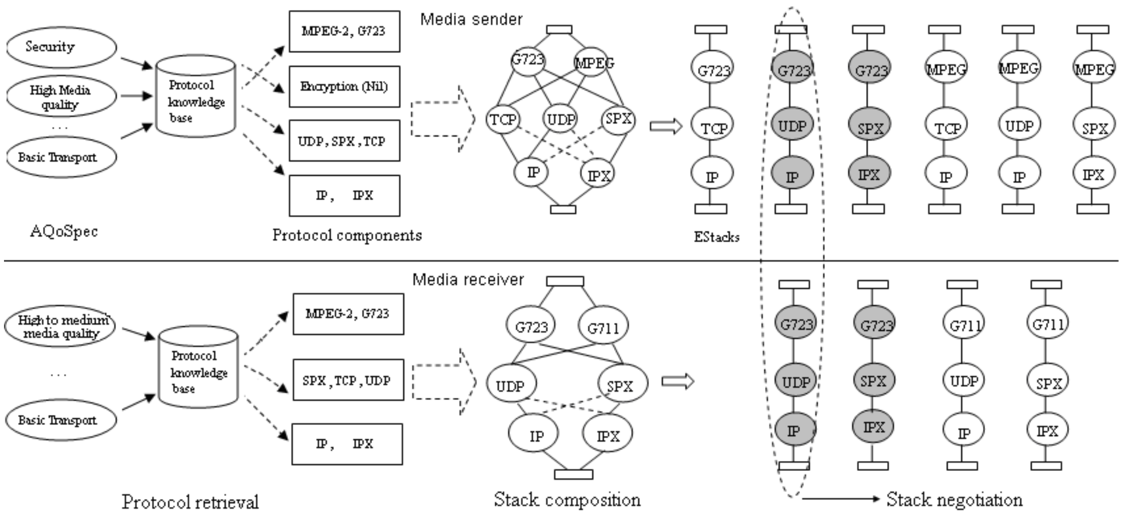


Figure 4-10: Semantic protocol selection and protocol stack building

On the other hand, not every combination of protocols will produce a valid stack. For example, the stack composition <JPEG, RTP, UDP, IPX> for video streaming is not acceptable because UDP is not compatible with IPX. We define a RDFS property element – *compatibility* – to capture such relationship among protocols in constituting a protocol stack (shown as dotted lines in the stack composition step of Figure 4-10). Similarly, the deployment of a protocol may rely on the adoption of other supporting protocols. For example, H263_RTP is a JMF implementation of standard video codec H.263 whose deployment requires the presence of RTP in the protocol stack. Such dependency can be described by another RDFS property element – *lowerDepd* – as the desired protocol RTP resides below H263_RTP in the protocol stack. Lastly, it is also possible that a protocol requires other protocols to appear on top of it in the stack. For example, TCP/UDP and IP are always bundled together in the protocol stack. Thus IP protocol has a dependency on upper side transport layer protocols (either TCP or UDP). We also introduce a property element – *upperDepd* – to address such a requirement (shown as solid lines in the stack composition step of Figure 4-10). RDFS definitions about these property elements can be found in Figure 4-11.

```
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xml:base="http://www.comp.nus.edu.sg/dpf/Protocol#”
>

< rdfs:Property rdf:ID="compatibility">
   <rdfs:domain rdf:resource="#Service"/>
   <rdfs:range rdf:resource="xsd:String"/>
   <rdf:property rdf:resource=”#transitive”/>
</rdfs:Property>

< rdfs:Property rdf:ID=" upperDepd">
   <rdfs:domain rdf:resource="#Service"/>
   <rdfs:range rdf:resource="xsd:String"/>
   <rdf:property rdf:resource=”#transitive”/>
   <dpf:depth rdf:resource=”xsd:Int”/>
</rdfs:Property>

< rdfs:Property rdf:ID=" lowerDepd">
   <rdfs:domain rdf:resource="#Service"/>
   <rdfs:range rdf:resource="xsd:String"/>
...
</rdfs:Property>
```

Figure 4-11: RDFS definition for compatibility and dependency

Compatibility and dependency are modeled as properties of the basic service class since all protocols may have such requirements. The dependency and compatibility knowledge of each protocol is to be supplied by protocol developers. To ensure that all protocols in the stack can interoperate with each other properly, stack validation is enforced once a stack is defined. The stack validation will go through two stages: *grammatical check* and *specification check*. The former guarantees that dependencies of every protocol and compatibility of the stack are satisfied. The latter makes sure that the stack is configured in accordance with user/application requirements. Those stacks that have passed the validation check are legal stacks and will be negotiated between communication peers for the final decision of stack composition and initiation of data transmission.

## 4.4 Network QoS Knowledge Modeling Briefing

As QoS support has not been part of the design considerations for virtually all network architectures, including that of the Internet, IETF has proposed several standard service models and mechanisms to support QoS delivery in the network since 1990th. These include, as said IntServ, DiffServ, MPLS and models that combine these technologies for an improved network scalability or utility. However, none of these models has now gained predominant acceptance to substitute current best-effort network service, which suggests that different QoS techniques may co-exist with QoS-transparent network to form a heterogeneous network QoS environment in the foreseeable future. From end-to-end QoS perspective, it is possible that a multimedia flow would traverse several autonomous network domains each employing its own QoS traffic models and service disciplines ranging from complex DiffServ to simple best-effort service.

Such a complex network QoS delivery situation has prompted our research initiatives in QCMF, which aims at designing an efficient end-to-end QoS framework coordinating heterogeneous QoS among end-host and network QoS components in a co-operative fashion. As explained earlier, each of such an administrative QoS network is modeled as an end-to-end QoS component in the context of QCMF. Here QCMF is not competing with any of the above standard network QoS techniques for a better performance in a local region; rather, our focus is on accommodating individual QoS techniques in each QoS component within QCMF by (1) modeling QoS characteristics of each end-to-end QoS component, (2) provide a platform to facilitate their information exchange and interaction, (3) enables a coordinated QoS decision-making in an end-to-end fashion in view of runtime QoS status.

At current stage, QCMF considers two network QoS scenarios. In a pure best-effort network environment, QCMF would perform end-to-end QoS provisioning leveraging on end-host QoS components (such as middleware QoS and application QoS). In a heterogeneous network QoS environment which consists of several network QoS components each adopting different QoS mechanisms, QCMF will model their QoS knowledge and capabilities such as traffic shaping policies at ingress/egress points, available service types, QoS parameters associated with each traffic class, current load information, packet dropping priority, and pricing strategies. Such knowledge will be fetched by end-hosts via signaling protocols so that an up-to-date picture about end-to-end QoS transmission is always accessible to end-hosts. Based on that, runtime QoS adaptations can be coordinated by end-hosts leveraging on a series of decision-making policies, heuristics and algorithms. Details about network QoS modeling and end-to-end coordination can be found in Chapter 6.

## 4.5 QoS Knowledge Processing

QoS knowledge processing traditionally focuses on the mathematical mapping, calculation and comparison of QoS parameter values. For example, in QoS negotiation, 10ms delay from one party is deemed worse than 5 ms delay from the other. Much work in this research area [8][69] has adopted such an approach, whereby a theoretic mathematical analysis can be found in [104].

QCMF provides numerical QoS parameter mapping and negotiation among communication peers in a way similar to other end-to-end architectures. (e.g., frame resolution negotiation, socket address selection). In addition to that, the knowledge plane established in QCMF specifically addresses the semantic processing of QoS information which is often neglected in relevant literatures (Figure 4-12). A knowledge plane in QCMF contains two essential functions: (a) semantic modeling of QoS knowledge of each end-to-end QoS component (which has just been explained), and (b) processing of such knowledge and sharing it among QoS components. An example of semantic mapping and composition of protocol stacks based on ontology/RDF has been shown in Section 4.3.3. Here we will discuss other essential QoS knowledge processing issues including knowledge sharing and first-order reasoning.

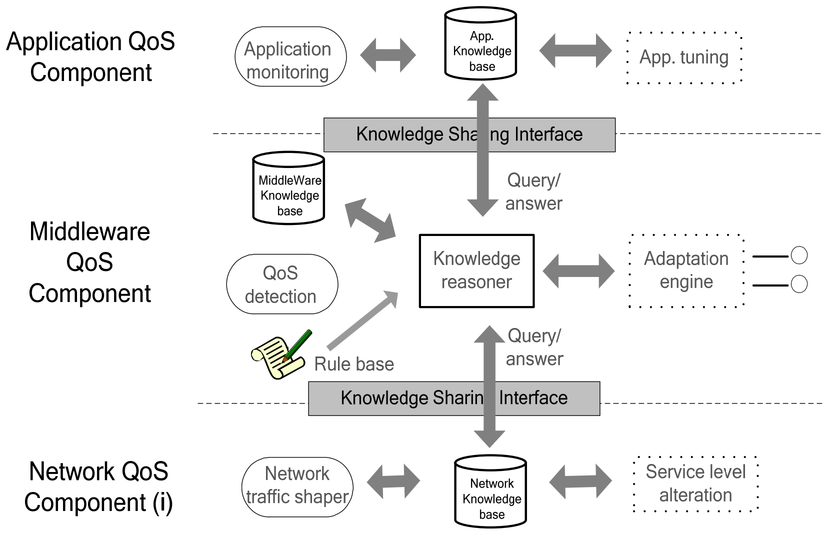


Figure 4-12: End-to-end QoS knowledge sharing and adaptation signaling

### 4.5.1 Knowledge Sharing

In QCMF, an end-host establishes three knowledge bases for information storage and sharing:

1) An application knowledge base typically contains applications QoS parameters and their relative adjustment priority, application specific configurations and adaptation

policies. Such knowledge is generally derived from application requirements input and domain ontology templates of applications.

2) A middleware knowledge base describes QoS capabilities (e.g., middleware adaptation choices) and restrictions (e.g., memory physical capacity) of a QoS middleware and end-hosts (e.g., device capacity). At the current stage of QCMF, focus has been put on the semantic description of QoS properties of communication protocols and composition of protocol stacks (as has been explained in Section 4.3).

3) A network knowledge base comprises network configuration and runtime conditions within each administrative network domain such as channel capacity, network service options, traffic load and pricing metrics.

In addition to the above static information, middleware knowledge base will also include dynamic information such as: (a) runtime monitored local information (e.g., current available bandwidth or current application performance) for the purpose of QoS analysis of transmission status; (b) the up-to-date QoS status of all network QoS components (shared from the network knowledge base; such information is also sharable with applications). The actual contents of each knowledge base are dependent on the implementation of a real end-to-end system. QCMF only defines the format of knowledge representation (i.e., ontology/RDFS as exemplified) to guarantee the machine readability and cross-domain inter-operability of QoS knowledge in heterogeneous end-to-end environments.

Knowledge sharing between various distributed QoS components in QCMF is fulfilled in both *push* and *pull* modes. The push mode is used for regular information exchange among QoS components. For example, static information about each QoS component is shared at session initiation stage. Runtime QoS status of a network QoS component is automatically updated with end-hosts once there is a significant change in its QoS status (e.g., QoS violation or network service re-distribution, details to be found in Chapter 6). Pull mode is used when an end-host QoS component is interested in the QoS information of other (remote end-host or network) components and hence is willing to learn such information. Knowledge sharing in pull mode is achieved via query and answer primitives (Figure 4-12), the generic definitions and usages of which is presented in Section 6.4.3. In our prototype implementation, we use Java *Remote Method Invocation* (RMI) technique [105] to setup and maintain the knowledge and control channel between QoS components. The main advantage of RMI is its implementation simplicity, interoperability between heterogeneous platforms and a certain degree of security been supported.

### 4.5.2 Knowledge Reasoning

Ontology/RDF-based knowledge modeling also enables a certain degree of QoS knowledge reasoning based on first-order logic, which can infer derived knowledge from known knowledge (i.e., static knowledge, profiled knowledge and monitored knowledge, refer back to Section 4.1.1 for details). Knowledge reasoning in QCMF comprises two forms: *ontology reasoning* and *user-defined reasoning*. Ontology reasoning supports all RDFS entailments described by the RDF Core Working Group. The main function of ontology reasoning is to check consistency among RDF/RDFS definitions when integrating or matching domain specific ontologies (e.g., AQoSpec to protocol stack composition).

| Properties | Ontology Reasoning Rules |
|---|---|
| subClassOf | (?A rdfs:subClassOf ?B), (?B rdfs:subClassOf ?C) -> (?A rdfs:subClassOf ?C) |
| type | (?P rdf:type ?A), (?A rdfs:range ?B) -> (?P rdfs:range ?B) |
| subPropertyOf | (?A rdfs:subPropertyOf ?B), (?B rdfs:subPropertyOf ?C) -> (?A rdfs:subPropertyOf ?C) |
| compatibleWith | (?A QCMF:compatibleWith ?B), (?B QCMF: compatibleWith ?C) -> (?A compatibleWith ?C) |
| lowerDepd | (?A QCMF:lowDepd ?B), (?B QCMF: lowDepd ?C) -> (?A QCMF:lowDepd ?C) |
| upperDepd | (?A QCMF: upperDepd?B), (?B QCMF: upperDepd?C) -> (?A QCMF:upperDepd?C) |

Table 4-2: Partial RDFS reasoning rule set in QCMF

Table 4-2 lists examples of RDFS reasoning rules in QCMF which is used to check the architecture integrity and correctness of QoS components (e.g., a protocol stack) at both build-time and runtime. The first rule shows that an RDF class definition is of transitive property while the second one indicates that a real instance of a RDF class automatically inherits all attributes from its RDF class definition.

In addition to standard ontology reasoning rules, QCMF also defines its own reasoning rules (i.e., user-defined reasoning) for QoS interpretation. Typical reasoning algorithms that can be incorporated include forward-chaining, backward-chaining and a hybrid execution model. The most common forward-chaining rule engine is based on the standard RETE algorithm [106]. The backward-chaining rule engine uses a logic programming engine similar to Prolog engines. A complex hybrid execution mode performs reasoning by combining both forward-chaining and backward-chaining engines. Rule 4 in Table 4-2 is a forward-chaining rule which specifies the compatibility relations among two QoS objects. Such a rule has been used in composition of a protocol stack that matches application requirements, as has been shown in Section 4.3.2. Meanwhile, it can be applied to the description and reasoning of other QoS objects of interest. For example, Linux and UNIX operating systems can be described as compatible systems with respect to packet scheduling (Figure 4-13), based on which the relationships among instances of different OS types can be identified systematically.

```
<rdf:RDF xmlsn:rdf=”http://www.w3.org/1999/02/22-rdf-syntax-ns#”
      xmlsn:rdfs=”http://www.w3.org/2000/01/rdf-schema#”
      xmlsn:OS=”http://lucan.ddns.comp.nus.edu.sg/QCMF/OS#”
>

<rdfs:Class rdf:ID="UNIX">
   <QCMF:compatibleWith rdf:resource="#Linux"/>
   <rdfs:comment data scheduling/>
</rdfs:Class>

<rdfs:Class rdf:ID="Linux">
   <rdfs:subClassOf rdf:resource="#UNIX"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Solaris">
   <rdfs:subClassOf rdf:resource="#UNIX"/>
</rdfs:Class>

<OS:Linux rdf:ID="RedHat">
       <description>instance</description>
       <version>8.0</version>
</OS:Linux>
```

Figure 4-13: Ontology definitions for some OS types and instances information

An example of backward-chaining rules is a trouble-shooting rule for diagnosing the root cause of a QoS violation, which is typically employed in a network QoS component. As is shown below, the consequence portion of the rule indicates the actions of either checking possible local conditions or forwarding the violation report to end-hosts for end-to-end QoS analysis.

- violation: (packetLost qcmf:violation TRUE) ->
(localDomain qcmf:fault linkFailure) $\vee$ (localDomain qcmf:fault routingCorrpution) $\vee$ (vioReport qcmf:forward ?QMan)

Other examples of forward-chaining reasoning in QCMF include QoS adaptation rules. Table 4-3 shows some reasoning rules for QoS adaptation defined in QCMF. As stated earlier, some adaptation rules are specific to application domains. The action of each rule may involve (a) a local adaptation to solve a violation within a QoS component, (b) a query been sent to other QoS components for information correlation, (c) an assertion of QoS status based on monitored end-to-end information, or (d) an adaptation coordination among QoS components for violation rectification.

| **Application Domains** | **User-defined Reasoning Rules** |
|---|---|
| A/V streaming QoS domain | network: hasSource(violation, Local) ^ network:status(packet_lost, HIGH) ^ network:status(delay, LOW) -> DivertViolation(middleware, vioReport) |
| | middleware: hasSource(violation, Network) ^ middleware:status(hasCost, Low) -> middleware:action(DPF, selective_retransmission) |
| Remote-tracking QoS domain | application:status(tracking_frequency, LOW) ^ application:status(object_velocity, MEDIUM) -> application:action(application, remove_trakcer) |
| Interactive audio QoS domain | network:status(throughtput, LOW) ^ middleware:status(CPU_load, LOW) ^ middleware: hasSource(violation, local) -> middleware:action (DPF, HIGH_COMPRESSION) |
| | network:status(throughtput, LOW) ^ middleware:status(CPU_load, HIGH) -> application:action(application, reduce_picSize) |
| | network:status(jitter, HIGH) ^ network: hasSource(violation, Local) ^ network:action(cost, HIGH) -> DivertViolation(middleware) |
| Distributed computing QoS domain | network:status(bandwidth, LOW) ^ application: hasSource(violation, network) application:status(mirrorServer, TRUE) -> application:action(application, handover) |

Table 4-3: Example first-order logic rules for coordinated QoS adaptation

## 4.6 SUMMARY

In this chapter, we address the knowledge modeling issue within the context of QCMF where a wide range of QoS-related knowledge comprising network level, middleware level and application level are defined and discussed. In tandem with the control plane which is intended for QoS monitoring and signaling among communication peers, the knowledge plane forms a basis for information sharing and decision-making with respect to QoS logics. A distributed approach for decision-making during QoS negotiation, configuration and adaptation is also highlighted to illustrate the necessity and advantages of our approach.

CHAPTER 5

# END-TO-END QOS VIOLATION ANALYSIS

Current researches tackle QoS violations through handcrafted if-then rules, which trigger corresponding actions once pre-described conditions are satisfied. The performance of such rigid rules is often not satisfactory in practice as will be discussed shortly. This chapter proposes an alternative approach to the diagnosis of QoS violations by monitoring and analyzing both end-to-end flow statistics and application performance.

## 5.1 DESIGN CONSIDERATIONS

Multimedia transmission over the network demands on end-host and network resources and requires in-time data delivery for real-time processing, failing to satisfy which will generally incur QoS violations. In view of this, various adaptive QoS systems have been proposed within end-hosts to best maintain transmission quality at runtime [107]. However, it is essential to identify the cause or nature of a QoS violation before an appropriate adaptation solution can be applied. An illustrating example is that the same phenomenon of packet loss in wireless communication can be due to one or more reasons (e.g., channel error or transmission congestion). Without a clear idea of the cause of an observed phenomenon, one may not be able to apply an appropriate remedy (e.g., error checking or rate control) toward the recovery of the transmission quality. However, existing researches usually ignore the detailed analysis of a QoS violation and instead equip their adaptive QoS systems with a few illustrating condition-action rules for the detection of and reaction to violations. Unfortunately, the results of such rigid rules are often not satisfactory in practice as will be discussed soon.

In fact, rule-based reasoning has been extensively employed in network management researches such as network trouble-shooting, performance monitoring and resource scheduling (e.g., [78][108][109][110][111]). Theoretical or empirical studies have been conducted in these literatures to demonstrate the correctness or effectiveness of their discrimination rules, policies or algorithms for automated interpretation of monitoring data in specific network environments. An example is [112] in which authors analyze dynamic routing information to detect and identify network disruptions.

End-to-end multimedia transmission has a distinctive two-tier logic in mapping application QoS requirements into network level performance metrics. Hence it becomes quite difficult to sustain QoS of multimedia applications in the presence of possible runtime violations: the causes of application QoS violations lie in low level service disruptions which is hard to be clearly identified and correlated. Unfortunately, most end-to-end QoS management systems have neglected the detailed analysis of QoS violations and focus more on other QoS topics such as service negotiation, resource allocation and QoS adaptation. These adaptive QoS systems often incorporate a few exampling condition-action rules to demonstrate their adaptation capabilities against runtime abnormity. As a result, these handcrafted adaptation rules which have seldom undergone serious (theoretic or empirical) studies may not yield satisfactory results (i.e., accurately detect QoS violations) in practice. We now analyze their limitations from the following two perspectives.

Firstly, existing work seldom explores the root cause of a QoS violation. Their justification of adaptation is often made on the partially observed phenomenon [113][114]. Among these studies, some focus purely on application behavior to identify the nature of a violation [115][116]. However, the same QoS degradation of multimedia applications could be caused by different reasons, each of which may be resolved in a unique way. Relying merely on application QoS metrics, we are unlikely able to differentiate among violations each with different nature. For example, an observation of inconsistent displaying of video frames in a video conferencing application may possibly due to the following causes: the sender machine is too busy to regularly generate frames; network is under severe congestion; the receiver application process lacks enough CPU time slice for video decoding. If the perceived jitter is caused by the CPU starvation problem of the decoding process, increscent of receiving buffer size (which is assumed in most literatures for solving jitter-related violations) surely will not help. Apart from these studies focusing primary on application behavior, a group of other researches examine flow level phenomenon for QoS violation identification [80][117]. For instance, some of them monitor flow level throughput and increase bandwidth allocation to the application if the detected throughput is under expectation [71]. However, the derivation of an application level QoS violation from a low level statement may not be accurate considering the complex QoS logic of multimedia applications. For instance, most video encoding formats use variable bit rate schemes which make it difficult to assert a violation from just the evidence of packet level throughput. All these examples suggest that different QoS violations may present similar observations in terms of one or a few parameters and hence differentiation is essential to find the actual cause of a violation.

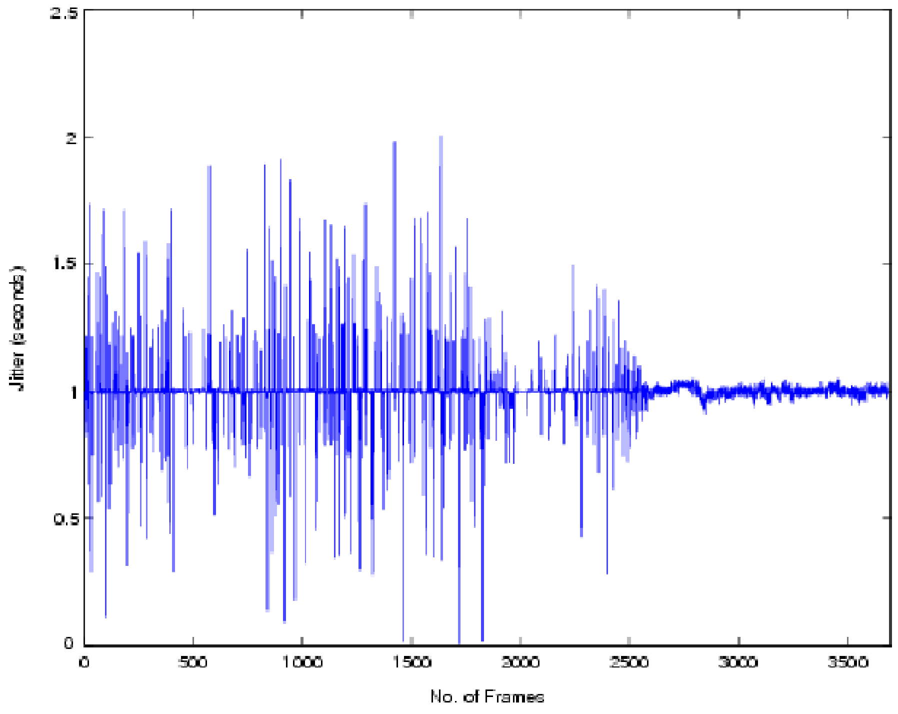


Figure 5-1: Observed jitter variation in a video transmission

Secondly, most current researches engage a rule/policy/algorithm based method to define responses to QoS violations (e.g., [108][118][119][120]). Violation thresholds are defined for QoS parameters in these rules. Much work is based on the assumption that the environmental impact on multimedia applications is well understood and hence can be expressed in simple mathematical formulas expressed in the form of rules. However, we found through experiments that it is not easy, if not impossible, to find a clear margin for each QoS parameter so as to correctly differentiate between different types of violations and a normal transmission. Figure 5-1 shows the packet jitter of a video streaming application during and after network congestion as we have observed in experiments (more details will be presented in section 7.3.1). It is obvious that packet jitter during congestion (appropriately the first 1500 frames) is statistically large and sparse compared with that in the subsequent non-congestion period (the last 2200 frames). Unfortunately, it is difficult to define a threshold jitter value to differentiate a congestion case (violation) from normal transmission. This is because some packets in normal transmission (e.g., frames 2300-2500) also exhibit large jitter values that is comparable to those transmitted in the previous congestion period. However, no QoS violation is observed at the application level in transmission and processing of frames 2300-2500 as will be shown later. On the other hand, we are not able to tell the cause of a violation by only checking packet jitter value (e.g., whether the observed abnormal transmission is due to sender busy or network congestion). To conclude, it may be insufficient (or even impossible) to confirm the appearance or the type of a QoS violation by merely examining single QoS parameter using a threshold-kind of approach. More comprehensive investigation into the characteristics of end-to-end QoS transmission is essential for the determination of QoS violations, as we have convinced through experimentation.

## 5.2 Overview of Our Approach

As discussed, the performance of existing rule-based QoS violation management is often not convincing in that no serious study has been done with respect to the relationship between the root cause of a violation and the observed violation phenomenon. In view of the above limitations, we propose a statistical classification approach to the QoS violation description and analysis. In particular, we hypothesize and validate through experiments that: (1) a QoS violation will present a consistent "fingerprint" in terms of the performance of the application and its end-to-end traffic pattern. Such a violation can be recognized once a similar fingerprint repeats during a transmission session, (2) QoS violations of different nature (e.g., caused by shortage of different resources) will exhibit diverse fingerprints. Using a set of end-to-end statistics, we are able to describe and differentiate between QoS violations.

Based on such a methodology, we have developed a machine learning approach to identify and differentiate QoS violations. Firstly, we monitor and gather traffic data of different types of violations. We examine application performance and correlate it with end-to-end packet flow information for a more holistic view of current transmission status. All these descriptors together depict the fingerprint of a violation, which is the manifestation of a fault in the end-to-end QoS system. Secondly, all violation data is fed into a learning machine for training. Runtime, the learning machine can recognize a QoS violation once real-time traffic data of similar nature is observed. Once the nature of a violation is confirmed, subsequent QoS adaptation which may take the form of either resource re-allocation or application behavior adjustment can be executed as per normal. Details of our approach will be presented later.

It should be noted that several recent literatures in QoS management have also reported the employment of machine learning algorithms for various purposes. Using video streaming datasets from a commercial server [121], Matthew Roughan [122] have defined four QoS classes (i.e., interactive, bulk transfer, streaming and transactional) and classify streaming traffic into one of these four categories using nearest neighbors and linear discriminant analysis algorithms. Similarly, authors of [123] have defined four QoS classes of different performance level for ATM network. Neural network (BP algorithm) is employed to sort traffic with different QoS requirements into one of these classes based on parameters such as cell loss rate and cell delay variation. A more up-to-date work following a similar track is [124], in which authors have proposed a QoS model for service differentiation in mobile ad hoc networks based on neural networks. They validate their models through simulation and show that by correctly assigning each kind of traffic to a proper service class, overall end-to-end performance (such as delay and throughput) can be improved. It should be noted that [68][71][123][124][125] classify QoS from performance perspective: applications of similar business nature and QoS requirements are grouped into the same category. In contrast, our work focuses on the identification of QoS violations through end-to-end observation and classification. [126] and a wealth of other studies differentiate the nature of packet loss in wireless communications. A lost packet can be identified as either due to wireless channel error or traffic congestion by examining packet inter-arrival time or relative one-way trip time. The determination of thresholds for these parameters is an engineering process similar to the way TCP protocol polishes its timeout value. These groups of studies focus specifically on the differentiation of wireless link error and network congestion. Comparatively, we propose in this paper a more general model to identify the causes of QoS violations for multimedia transmission in the Internet. Closer to our work is [115] which has also mentioned the importance and complexity of end-to-end QoS violation diagnosis. They have designed a QoS management architecture that instruments an application for violation detection. Such a system continuously adjusts resource allocation to multimedia applications until their QoS requirements are fulfilled. In this way, QoS violations detection rules and thresholds can be derived through experimentation [127]. Comparatively, our work focuses on identifying a QoS violation through comprehensive observation of end-to-end QoS and traffic pattern. To the best of our knowledge, our work is the first one to statistically deal with end-to-end QoS violation diagnosis for multimedia transmission.

## 5.3 End-to-end QoS Violation Analysis

Essentially, a few issues need to be addressed to make an adaptive QoS system effective in coping with QoS violations. These include (1) the clear-cut definitions, (2) effective detection, and (3) timely response to QoS violations. We will address these issues step by step in the follow sections. The first step in our QoS violation analysis is to assert the existence of a violation (through examining application performance) and then annotate it using a set of end-to-end QoS parameters. To achieve this, we take a top-down approach in the modeling of end-to-end QoS violations.

### 5.3.1 End-to-end Monitoring of QoS Violations

We describe a QoS violation using monitoring data collected end-to-end, which includes both application level QoS parameters and flow level metrics for a more complete review of a violation. To collect these performance metrics, software monitors are placed at different layers of the media receiver. In the following sections, we explain our approach to QoS violation analysis through an example of video transmission that we have designed and used in experiments. Such a video flow can be either lively captured from a web camera or fetched from a media file on the hard disk of the media sender. The original video flow is transcoded into RTP compatible formats (e.g., MPEG_RTP format) and encapsulated into RTP packets for streaming[5]. The media source constantly produces video frames at the rate of 30 frames per second (fps). Due to various interferences on the end-to-end path, the receiving and displaying frame rate at the media sink may deviate significantly from the sending rate and hence gives rise to our violation analysis.

[5] The Real-time Transport Protocol (RTP) is a packetization standard for delivering multimedia (e.g., audio and video) contents over the Internet and is defined in RFC 3550 [59].

### 5.3.2 Application QoS Violation Indicator

| QoS dimensions | Options |
|---|---|
| Frame rate (fps) | 30, 25, 20, … |
| Resolution (pixels) | 656*272, 352*266, 176*144, … |
| Color depth (bits) | 32, 16, 8, … |
| A/V synchronization (app. specific) | High, Medium, Low |
| Packet loss ratio (app. specific) | High, Medium, Low |

Table 5-1: Tunable parameters in video transmission, applications

An adaptive QoS system aims at detecting and reacting to runtime quality degradation to the best of its ability. The identification of QoS violations in such a system hence should project on the performance variation of multimedia applications. In view of this, we use application QoS metrics to assert the appearance of a QoS violation: if the observed application QoS is below expectation, a violation is said to happen.

Most multimedia applications in practice would have more than one QoS parameters, the entirety of which constitutes the application QoS. For example, a video transmission application (such as the one developed and used in our experiments, details to be presented in Section 7.3.1) typically has several tunable QoS parameters as listed in (Table 5-1). These parameters may compete on the same set of system resources for their respective performance. For instance, the performance of both picture resolution and color depth relies on end-host resources (e.g., CPU time slice) and network resources (e.g., link bandwidth). In many application scenarios, there exists a most important QoS parameter to sustain, whereby other parameters can be traded off in case of insufficient resources (e.g., [8]). We define such a QoS parameter to be best assured as a principal QoS parameter, failing to sustain which indicates the existence of a QoS violation. In this sense, such a principal QoS parameter serves as an *application level indicator for QoS violations*. In other cases where graceful QoS degradation is desired among all QoS dimensions, Quality of Perception (QoP) investigations can be conducted as in those relevant literatures (e.g., [48][128]) which derive the application QoS violation indicator as the weighted sum of all QoS parameters. In either case, the application QoS violation indicator marks out the appearance of an application QoS violation which triggers our end-to-end QoS violation diagnosis.

To focus more on our discussion, we use "video frame rate" as an example application QoS violation indicator in our video transmission experiment[6]. As said, the media sender constantly produces video frames at 30fps during experiments (while the flow bit rate is varying). Due to various interferences or impairments along the end-to-end path, the receiving and displaying frame rate may deviate significantly from the sending rate. Once the video frame rate has dropped below user expectation, a QoS violation is asserted to happen. Corresponding flow level performance metrics will subsequently be collected for analysis through software monitors placed at the media receiver.

In line with other relevant studies [96], we set the boundary value of video frame rate to 25fps, below which signifies the appearance of a QoS violation. Upon that, flow level analysis will be executed to find out the nature of that violation (e.g., frame jitter due to server busy or network congestion). We sample the video transmission application every 2 seconds to check for the displaying frame rate at the media sink. The selection of this sampling interval is a tuning process during experiments that seeks the balance between measurement accuracy and overhead: higher monitoring

[6] As can be easily found out, other single QoS parameter or a weighted QoP value can be engaged in the same way.

accuracy demands more frequent sampling which in turn, consumes more resources (e.g., CPU cycle) and hence threatens the performance of the target application [10].

### 5.3.3 Correlate Application QoS Violations with Flow Statistics

Once a QoS violation is asserted through examining the application QoS violation indicator, corresponding flow information will be collected for a complete review of current end-to-end QoS status. As explained before, a QoS violation will present a consistent fingerprint in terms of the observed end-to-end application behavior and flow traffic pattern. Meanwhile, different violations will present diverse end-to-end QoS fingerprints, which also differ from that of a normal transmission. We use flow descriptors listed in Table 5-2 to portrait flow characteristics of a QoS violation. These descriptors together with application QoS parameters, can well describe a QoS violation as is to be shown in the experiments.

| **Flow descriptors** | **Remark** |
|---|---|
| Packet delay | Average packet delay in a frame |
| Delay jitter | Average packet jitter in a frame |
| Packet loss | Total packet loss in a frame |
| Receiving throughput | Calculated for each sampling interval |
| Other metrics | Specific to individual QoS domains |

Table 5-2: Flow descriptors for end-to-end QoS

A real-time QoS management system should detect and react to QoS violations in the magnitude of seconds. In view of this, the sampling interval for violation detection is commonly set to a few seconds (we set the interval to 2s as explained). A media receiver which processes approximately 30 video frames per second will receive a few hundred of packets that carry media contents. These packets are typically of similar size except the last one of each media frame. Using these facts, we can estimate that hundreds of (or even a few thousand of) packets will reach and be processed by the media receiver within one sampling interval. Once a violation is asserted through checking the application QoS violation indicator, there arises a problem as how to efficiently discover those packets (among hundreds or thousands of packets) which most precisely depict the fingerprint of the current QoS violation.

We correlate an application QoS violation with flow manifestation using *RTP timestamp* and *sequence number*. In RTP transmission, the sequence number increases by one for each RTP packet been sent and hence can be used by the receiver to detect packet loss or restore packet sequence. The timestamp of each RTP packet, on the other hand, reflects the sampling instant of the first octet in the RTP data packet at media sender. As a result, RTP packets that carry payload of the same media frame will share the same timestamp.

Assuming that we have received a trace of (sorted) RTP packets, namely $p_1, p_2,...p_n$ during two sampling points $t_{k-1}$ and $t_k$. These packets are assembled into $m$ frames $f_1, f_2,...f_m$ for decoding and display. A QoS violation is ascertained at time $t_k$ by digesting the average frame rate during the last sampling interval. We find representatives of those "violated" frames as follows:

- We identify all member RTP packets of the first and last frames $f_1$ and $f_m$ received during this interval based on the RTP timestamp. Among the rest, we randomly choose a frame, say $f_i$ and also examine its member RTP packets.
- We compute flow descriptors for frames $f_1$, $f_i$ and $f_m$. Average packet delay $d_i$ and jitter $j_i$ for the $i^{th}$ frame are calculated as mean values of delay and jitter of

all member RTP packets (denoted as $d_p$ and $j_p$) of that frame[7]. Throughput is measured as the packet arriving rate in the sampling interval. For example, the average frame delay and jitter for frame $f_i$ are calculated based on the following formulas:

$d_i = (d_{i,1} + d_{i,2} + ...d_{i,k}) / k$ ,

$j_i = (j_{i,1} + j_{i,2} + ...j_{i,k}) / k$ ,

for all RTP packets that have $TStamp_{p_t} = TStamp_{p_{t,1}} = ... = TStamp_{p_{t,k}} = TStamp_{f_i}$ .

- For each candidate frame (i.e., $f_1$, $f_j$ and $f_m$), flow descriptors and application QoS parameters (e.g., frame rate, resolution) are sent to the violation classification algorithm for analysis (see next section). If all the three tests conclude the same type of violation, a QoS violation is confirmed and corresponding adaptation action can be triggered.

The approach of our flow summarization and violation analysis is designed in light of the following considerations. Firstly, we are not obligated to investigate into all media frames of a sampling interval for violation justification, which is much more time consuming. Hence we choose three "typical" delegate frames for violation analysis. Secondly, for each delegate frame, we calculate average statistics covering all member RTP packets of that frame for a comprehensive representation of frame QoS statistics. Lastly, we require the identical results from three testing to acknowledge a violation. Otherwise, the algorithm will advance to the next round of examination. As will be shown in Section 7.3.2, the classification accuracy for each test is between 86%-97% in those environments we have tested. Hence, if all three tests hit the same results, statistically we have a good chance to assure the nature of a QoS violation.

[7] The computation is done on correctly received and identified packets only. Those lost or corrupted packets are not counted in.

Our approach requires the continuous monitoring of a RTP flow so as to correlate a violation with flow statistics. We achieve this by listening to the network interface card (NIC) of the media receiver. We insert a sending packet timestamp in each IP packet (distinguishable from the RTP frame timestamp for frame re-construction) so that traffic statistics of the IP packet can be calculated once it reaches the receiver host. We have found through experiments that the overhead to log RTP packets is not noticeable to the performance of a multimedia application since we have designed it as a system-level utility. However, in those environments that require minimal resource occupation (e.g., mobile devices such as handsets), selective sampling techniques (e.g., [129]) can be engaged. The violation analysis algorithm listed in this section can be applied with comparable performance considering the statistical nature of our method.

## 5.4 VIOLATION CLASSIFICATION WITH NEURAL NETWORK

We have engaged several neural network algorithms for the identification and classification of QoS violations. Once a QoS violation is asserted runtime and end-to-end statistics are collected, relevant data will be fed into the neural network for the identification of the nature of a violation. This section briefly discusses the various neural network algorithms we have employed for QoS violation classification, including the one proposed by us for a fast learning and classification.

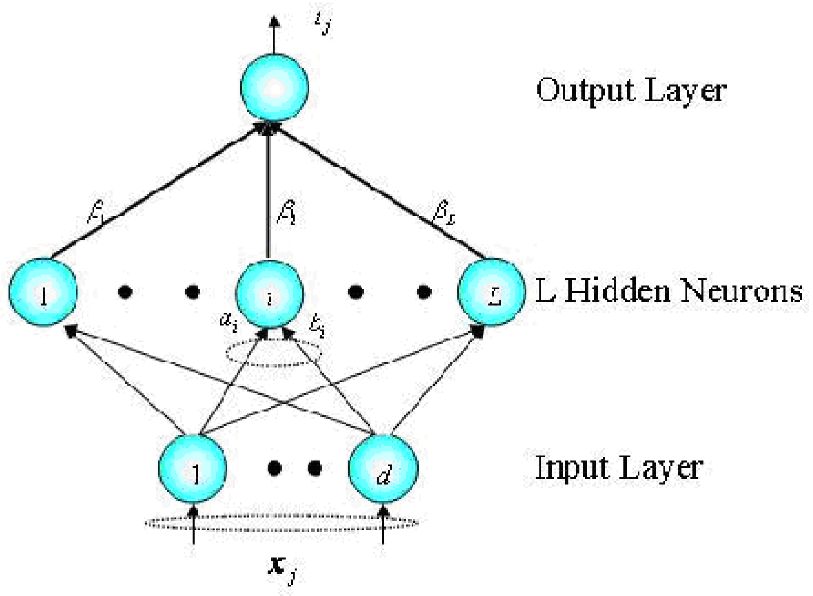


Figure 5-2: Single Hidden Layer Feedforward Neural Networks

### 5.4.1 Neural Network Algorithms Briefing

Neural networks have been successfully applied to a wide range of real world classification problems in the past decades [130]. Figure 5-2 is a standard single hidden-layer feedforward neural network (SLFN) model. The output of a SLFN with $L$ hidden neurons can be represented by $f_L(X)=\sum_{i=1}^{L}\beta_i g(a_i,b_i,X)$, where $a_i$ and $b_i$ are the learning parameters of the $i^{th}$ hidden neurons, and $\beta_i$ is the weight connecting the $i^{th}$ hidden neuron to the output neuron (also called the hidden-to-output weight). $g(a_i,b_i,X)$ is the output of the $i^{th}$ hidden neuron with respect to input $X$. Based on different parameter combinations of the activation functions, two main neural network models has been widely used in SLFN: for additive neurons with activation function $g$, $g$ is defined as $g(X)=g(a_i \cdot X+b_i), a_i \in R^d, b_i \in R$, where $a_i \cdot X$ denotes the inner product of vectors $a_i$ and $X$ in $R^d$; for RBF (radial basis function) hidden neurons with activation function $g$, $g$ is defined as $g(X) = g(\frac{\|X - a_i\|}{b_i}), a_i \in R^d, b_i \in R^+$, where $R^+$ indicates the set of all positive real value.

In applying neural network algorithms to the classification of QoS violations, the input vector $X$ consists of flow parameters such as packet delay, jitter and frame rate that we have runtime collected. The target vector $t$, which indicates the classification result, shows the category a violation belongs to. The configuration of other internal calculating vectors and matrixes will be explained in Section 7.3.

In search of suitable neural network algorithms for QoS violation classification, our focus has been put on their respective learning speed. This is due to a special design consideration of our end-to-end QoS management system: the initially available training data which is (1) provided by the system designer (or network administrator), or (2) collected through a few online testing clips, may not be accurate (or complete) enough to effectively differentiate all QoS violations. In view of this, an execution-feedback process is proposed to “refine” the neural network structure for a more precise violation snapshot: the end-to-end QoS system is firstly trained with the initial data; once a QoS violation is detected, a predefined adaptation will be invoked according to the analysis result; the effect of the adaptation is then observed, which in turn verifies whether a QoS violation has been correctly identified and solved. After such a cycle, fresh violation data (i.e., monitored data of the violation that has just been confirmed and solved) will be treated as new training data and learned by the neural network algorithm. In this way, QCMF will update its violation profiles and becomes more and more accurate in identification of QoS violations as the transmission goes on. As can be found out, our QoS management system requires a fast neural network algorithm for online QoS violation learning [131]. Hence, we group and analyze those algorithms we have engaged during experiments into *offline* and *online* categories according to their training speed.

### 5.4.2 Offline Algorithms

- *RANEKF*. An important sequential learning algorithm of RBF networks is proposed by Platt through the development of a resource allocation network (RAN) [132]. RANEKF [133], which is known as an enhancement of RAN, use an extended Kalman filter to update the neural network parameters so as to improve the accuracy and achieve a more compact network architecture.

- *MRAN*. RANEKF can only add neurons to the network and cannot prune the insignificant neurons from the network. A major performance improvement of RANEKF is achieved in [134] by introducing pruning strategy based on the relative contribution of each neuron.

- *Levenberg-Marquardt BP (LMBP)*. LMBP [135] is adjusted from Newton's method through the approximate Hessian matrix. As has been recognized, Newton's method is faster and more accurate near an error minimum. In view of this property, the objective of LMBP is to shift towards Newton's method as quick as possible. So far LMBP has been known as one of the fastest gradient based learning algorithms.

### 5.4.3 Online Algorithms

- *Stochastic BP*. In stochastic BP [136], weights may move down along the random gradient in each iteration. Therefore, it may result in a better classification solution since it avoids the probability of achieving local minimum. Due to its random characteristic, stochastic BP can provide good generalization performance in a short time.

- *Extreme Learning Machine (ELM)*. Huang proposed a simple learning algorithm for SLFN called ELM [137] which randomly chooses hidden neurons and analytically determines output weights by using generalized inverse matrix. Neural networks with ELM can provide good generalization performance at quite fast learning speed.

- *Orthonormal algorithm*. In Appendix A, we will prove that an orthonormal network is universal approximation and hence can be used for classification. In addition, we will extend the scope of hidden neurons from kernel functions to additive functions, the latter of which is commonly known of higher classification accuracy [138]. Our orthonormal algorithm for QoS violation classification is also presented in Appendix A.

A major concern that we employ neural network algorithms to replace traditional rule-based methods in QoS violation identification is that the former approach enjoys much more flexibility. For example, neural networks can automatically find the weight of and relationship among each contributing parameter. Furthermore, no clear margin of a QoS parameter is needed to identify the type of a violation owe to its statistical nature. However, conventional neural networks (e.g., offline algorithms as listed above) have heavy computational burden involved in solving highly complex functions with a large number of variables, which will lead to a long training time. As fast QoS violation training and identification is crucial in runtime QoS management, online algorithms which exhibits quick response are more in favor, as we will demonstrate in Section

7.3.2. Those offline algorithms engaged in our experiments are mainly for comparison purpose.

## 5.5 SUMMARY

Common approaches to QoS violation analysis involve hypothesis to violation identification through checking threshold(s) of one or a few QoS parameters. They often ignore the fact that different QoS violations may present similar phenomenon in the observation of one or a few QoS parameters. We argue that more comprehensive survey is needed to recognize the cause of a violation so that a correct adaptation remedy can be applied. In this chapter, we propose a statistical approach to QoS violation analysis which involves (1) violation assertion through inspection of the application QoS violation indicator, (2) violation modeling through a collection of application performance and flow statistics, and (3) violation classification using fast machine learning algorithms. The performance of our statistical approach to QoS violation identification will be introduced in the experiment sections.

# CHAPTER 6 CROSS-COMPONENT QOS ADAPTATION

Meeting performance requirements of QoS-sensitive applications is fundamentally an end-to-end issue, which requires all QoS-enabled facilities along the end-to-end path to work cohesively to achieve the desired end-to-end effect. However, most of the existing QoS solutions focus on their respective areas while paying little attention to the interaction of other QoS facilities on the end-to-end path. As a result, QoS can be sustained in their local domains while no satisfactory end-to-end performance can be provided to end-users. This chapter presents the cross-component QoS coordination scheme in QCMF, whereby QoS adaptation assumptions, strategies, heuristics and algorithms are discussed in detail.

## 6.1 END-TO-END QOS MODEL

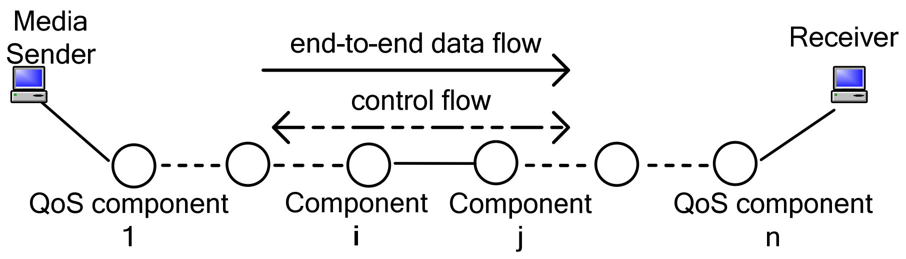


Figure 6-1: Abstracted end-to-end QoS provisioning model

An end-to-end QoS framework is a management system for organizing all QoS mechanisms along the end-to-end path. We abstract the end-to-end QoS provisioning model in Figure 6-1 in which each QoS network administrated by an Internet Service Provider (ISP) is represented as a bubble. Compared with the QoS reference model introduced in Section 3.1, Figure 6-1 provides a more high level yet comprehensive overview of end-to-end QoS whereby (1) diverse QoS facilities in end-hosts (e.g., QoS-enabled OS, middleware or application) are summarized and represented by a single end-host QoS architecture in the overall picture of end-to-end QoS provisioning, (2) autonomous networks of different QoS characteristics are modeled respectively as end-to-end QoS components. These networks interconnect with each other and form the end-to-end communication path for packet delivery. It is possible that each network may employ distinctive QoS mechanisms in its local region. For example, some ISPs may adopt DiffServ QoS model while others may choose MPLS tunneling technique. Engaging the same DiffServ techniques, different network domains may further define domain-specific configurations and policies. Irrespective of these implementation details, we denote each autonomous network that has specific QoS mechanisms as a network *QoS component* in our end-to-end QoS provisioning scenario.

In QCMF, end-to-end QoS delivery is decomposed into two hierarchical levels: *intra-domain provision* and *inter-domain collaboration*. The former enforces QoS delivery within a QoS component via management functions such as resource allocation and scheduling while the latter manages end-to-end QoS by dividing global QoS objectives into QoS commitment of each QoS component.

When an end-to-end transmission session is established, QoS parameters are negotiated among all QoS components along the end-to-end path. A consistent service is thus provided to an end-user in which the overall QoS objective is divided into service contracts with each end-to-end QoS component. For example, the end-to-end latency requirement of a multimedia application can be decomposed into specification of transmission delay in each network QoS component and processing delays in end-hosts, such that their sum equals or is less than the end-to-end delay constraint. A negotiation protocol (e.g., RSVP or [29]) can be engaged to reach such service level agreements (SLAs) among end-to-end QoS components.

Runtime, each QoS component will try to meet the obligation of its SLA specification. However, QoS violations may occur within a QoS component for various reasons such as temporary traffic overload, hardware failure of a router or routing cycle. QoS violation handling in QCMF is correspondingly divided into two hierarchical levels: a violation can be tackled by QoS adaptation within the QoS component where the violation has occurred (known as the source QoS component of a violation, $C_{vio}$) or through coordination and adaptation at another QoS component whose adaptation can compensate for the QoS degradation at $C_{vio}$.

For example, delay/jitter violation at QoS component $i$ can be solved by local service upgrade, compensated by QoS upgrade at QoS component $j$, or by data buffering and/or timeout/retransmission mechanisms at the flow receiver. As discussed in Section 2.2.1, the last solution is often not a sustainable one, but can still provide certain degree of QoS preservation in the case that other end-to-end adaptation choices are not applicable. On the other hand, a bandwidth starvation problem can be resolved by either increasing bandwidth allocation in the network or by adopting a higher compression rate codec at the media sender so that less bandwidth is to be consumed by the media flow. In the worst case where QoS is no longer sustainable no matter what kind of end-to-end adaptation is enforced, QoS component handover is necessary which replaces $C_{vio}$ with another candidate component of better QoS services[8] (e.g., [72]). For instance, a DiffServ subnet in the end-to-end path that suffers from severe traffic congestion can be substituted by another one of shorter transmission delay so that the overall end-to-end latency can still be kept within user expectation. However, the dynamic composition of end-to-end provisioning path is technically more complex (e.g., depending on the availability of routing topology) and time-consuming. To focus more on our research, we restrict our discussion to the scenario where runtime resource fluctuation and QoS violations can be sufficiently solved within current end-to-end QoS settings through end-to-end coordination and adaptation and hence no handover of QoS components is needed. Particularly, we concentrate on the end-to-end delay/jitter violation and show how QCMF solves it through runtime coordination and adaptation. We will first describe our models for both network QoS and end-host QoS. After that, we discuss the end-to-end evaluation and adaptation heuristics and algorithms in Section 6.4.

| **Service Class** | **Delay Performance (s)** | **Bandwidth Capacity (Mbps)** | **Unit Price (/hour/Mbps)** | **Available Bandwidth (Mbps)** |
|---|---|---|---|---|
| 1. Predictive | 0.5 | 100 | \$2 | 70 |
| 2. Predictive | 5 | 300 | \$0.8 | 10 |
| 3. Predictive | 15 | 450 | \$0.25 | 200 |
| 4. Best Effort | Unspecified | 150 | - | 60 |

Table 6-1: Service options table of a network QoS component

## 6.2 NETWORK QOS MODEL

From the end-to-end perspective, each network QoS component on the provisioning path appears as a service provider which allocates local resources to guarantee the

[8] Such a handover relies on the possible alternative routes between the media sender and receiver. Hence, a routing algorithm or protocol should be engaged together with our scheme to dynamically figure out the network topology.

performance of an end-to-end flow within the scope of its operation. Typically a QoS service provider can offer its service at different levels of quality to a service requester (i.e., an end-to-end flow), some of which are better than others in terms of QoS performance. We denote a service class $h$ offered by network QoS component $i$ as $S_{i,h}$ . $S_{i,h}(h=1,2,3...)$ constitutes a *service degradation path* which lists all possible service options of that QoS component. As is shown in the example *service options table* (Table 6-1), a network QoS component can offer four service classes to an end-to-end flow, each with different local QoS support (e.g., delay assurance) at different subscription cost. On the other hand, each service class has limited bandwidth capacity and hence can accommodate limited number of users. A practical example of a network QoS component with multiple service levels is the DiffServ model which defines three service classes for the differentiation of traffic nature and routing of packets. However, our definition and discussion of service class and service degradation here is more general (i.e., we do not endorse specific meanings for each service class) and is purely from end-to-end QoS perspective, which is different from the design initiatives of DiffServ (e.g., scalability and utility of networks).

The definition of service class and service degradation path allows a network QoS component to abstract and expose its status and resource availability to an end-to-end flow in a manageable and tractable manner so that a coordinated end-to-end adaptation becomes possible. Suppose that an end-to-end flow subscribes to service class 2 of a network QoS component as illustrated in Table 6-1, which currently undergoes a local delay violation due to temporary network congestion. A local QoS adaptation can be executed (through an request from the flow sender/receiver) which upgrades the flow to service class 1 of better QoS support so as to eliminate such a delay QoS violation. Alternatively, QoS adaptation may take place in another network QoS component (possibly also through local service upgrade) so that end-to-end delay is still kept within the acceptable range. Such coordinated end-to-end adaptations require each network QoS component to define and manage (e.g., update with end-hosts) its service options table so that a correct choice could be identified with respect to where to issue the adaptation request and how to do the adaptation. Details about the decision-making process are to be elaborated in Section 6.4.3.

For the benefit of end-to-end coordination and adaptation, we require each network QoS component to (1) monitor the service status of its service classes, and (2) notify the flow receiver the appearance of a local violation which has just happened in the service class it has subscribed. The first task can be fulfilled capitalizing on existing monitoring facilities of an autonomous network. For example, DiffServ employs network monitoring facilities to understand updated network load in each service class for the purpose of admission control of a new flow. On the other hand, an end-to-end flow needs to sign a SLA before it can enjoy any service from a network QoS component. Hence the second requirement (QoS violation notification) can be satisfied by a network QoS component (e.g., through a bandwidth broker in the DiffServ case) which has the addresses of all end-hosts in the SLAs. Such a notification is intended for end-hosts to double-confirm the appearance of a QoS violation (recall that the flow receiver can identify a violation through end-to-end monitoring flow traffic as introduced in Chapter 5). We assume that all the control messages are transmitted in an out-of-band mode and hence are not affected by the appearance of a QoS violation. It should be noted that we do not require a network QoS component to monitor the performance of an individual end-to-end flow; instead it needs to watch the service status of each service class. Hence, such a scheme will not cause any scalability issue in network management.

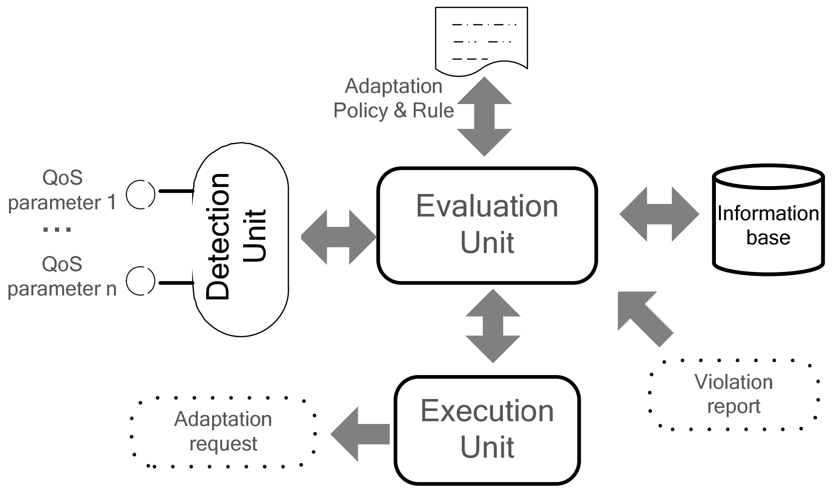


Figure 6-2: End-host QoS management model (QMan)

## 6.3 End-host QoS Model

In our QCMF framework, end-hosts are responsible for the overall management of end-to-end QoS. These include tasks such as QoS negotiation, which is conducted at session initiation stage to sign service contracts with network QoS components, and QoS adaptation, which is enforced at runtime to best maintain transmission quality against possible violations. A QMan middleware has been developed within QCMF to fulfill these tasks, as has been introduced in Section 3.3.

As mentioned earlier, QoS adaptation can be performed either at the QoS component where the violation happens (i.e., $C_{vio}$ ) or at another QoS component that can compensate for the service degradation at $C_{vio}$. In QCMF, such an adaptation selection is done by the QMan middleware at the receiving end-host which has the information of all end-to-end QoS components. The logical model of QMan is illustrated in Figure 6-2. Upon the appearance of a QoS violation, an *evaluation unit* is responsible for selecting an end-to-end QoS component best suitable for QoS adaptation. Such a decision is made based on the information exchanged in the knowledge plane (Chapter

4). Once an adaptation decision is made, an *execution unit* communicates with the selected QoS component for adaptation control and signaling. Finally, a *detection unit* (1) continuously monitors the performance of end-to-end flow to detect a QoS violation (Chapter 5), and (2) evaluates the effect of QoS adaptation for validation of violation problem-solving.

A prevailing requirement of end-to-end QoS adaptation is to timely recover multimedia transmission quality. It suggests that a complex end-to-end framework that seeks a globally optimal solution may not be applicable in a real-time transmission scenario due to its long convergence period. Hence QCMF aims at identifying an “acceptable” adaptation solution, which can quickly bring end-to-end QoS back to a normal state in a short time. Please note that the normal state after adaptation may not be the same as the initial state of QoS delivery but is acceptable to the performance of a multimedia application. Hence the result of such an adaptation strategy may not be end-to-end optimal but is effective in that it can quickly restore the transmission quality. On the other hand, a QoS framework employing fine-grained QoS control over all the end-to-end QoS components may not yield the desired effect in a distributed end-to-end scenario due to its stability problem and validity in the context of possible tussles and intricate entanglements among various QoS factors [139]. In view of this, QCMF does not care about the implementation details and inner behaviors of each QoS component, but aims at designing a platform for abstracting those QoS features of a QoS component that are pertinent to end-to-end coordination. For instance, a DiffServ subnet may engage traffic policing techniques to regulate the performance of packet flows of an overloaded service class. Therefore a minor local network congestion violation could be decently solved within the subnet before any end-to-end QoS adaptation action is raised. In this sense, a two layer QoS adaptation scheme is designed in QCMF as said, namely a rapid local adaptation (done by individual QoS components through self-configuration) and an end-to-end re-scheduling (achieved through end-to-end analysis and coordination). While the former adjustment may not be noticeable to the end-hosts, QCMF concentrate on the latter end-to-end QoS adaptations that target at severe QoS violations beyond the ability of local adaptation. Bearing the above design considerations in mind, we have designed a set of adaptation algorithms within the QCMF framework, which is to be elaborated in the next section.

| **Service Class** | **Service Model** | **Capacity (Mbps)** | **Unit Price (/hour)** | **Historical record 1** | **Historical record 2** |
|---|---|---|---|---|---|
| 1. Predictive | $\lambda_1, \tau_1$ | 100 | \$2 | 80Mbps @ 2007-3-15 17:42 | 70Mbps @ 2007-3-15 17:27 |
| 2. Predictive | $\lambda_2, \tau_2$ | 300 | \$0.8 | 200Mbps @ 2007-3-15 17:37 | 170Mbps @ 2007-3-15 17:17 |
| 3. Predictive | $\lambda_3, \tau_3$ | 450 | \$0.25 | 300Mbps @ 2007-3-15 17:49 | 330Mbps @ 2007-3-15 17:42 |
| 4. Best Effort | - | 150 | - | - | - |

Table 6-2: Service status table of a network QoS component as is maintained by QMan middleware inside the flow receiver; for each network QoS component, a corresponding table is kept by QMan and updated through either push or pull mode

## 6.4 End-To-End Coordination and Adaptation

In this section, we introduce several algorithms that run in end-hosts for a coordinated end-to-end QoS adaptation. These include an information gathering algorithm which keeps the media receiver updated with the QoS status of each network QoS component, an evaluation algorithm which reviews all possible choices and selects a QoS component for adaptation, and finally an execution algorithm, which uses defined primitives for adaptation control, signaling and feedback. As stated, these algorithms are executed by QMan middleware at the receiving end-host in a pure end-to-end manner.

### 6.4.1 Information Gathering Algorithm

The main function of the information gathering algorithm is to update QMan middleware QoS information of each network QoS component so that a correct adaptation solution can be selected upon the appearance of a QoS violation. For each network QoS component, QMan keeps a *service status table* that describes QoS meta-data of that component as shown in Table 6-2. QoS Information to be recorded can be summarized into static and dynamic categories. The former refers to the service information of a network QoS component (e.g., the number of service classes, capacity of each class and unit price for purchase), which is advertised by a network QoS component in its service options table. The latter refers to the dynamic workload information of a network QoS component (e.g., 80 out of 100Mpbs bandwidth of a service class have been subscribed at the moment the end-to-end flow logs in), which is changing according to the traffic load of a network.

The information gathering algorithm is executed at two stages. At session initiation stage where QoS negotiation is performed, each QoS component reports its service information (i.e., static information) and current workload information (i.e., dynamic information) to the flow receiver. At runtime, a network QoS component will update its current working status and spare capacity to QMan once there is a major update. The definition of a major update refers to (a) a local violation due to reasons such as temporary traffic congestion, software crash or equipment failure, (b) rejection or confirmation of an adaptation request from the flow receiver, (c) regular updates of QoS information based on a pre-negotiated timeslot. The first kind of QoS update refers to the situation that a network QoS component reports its local QoS violation to flow receivers that have subscribed the service (which is already explained in Section 6.2). Current QoS status of that network QoS component will be appended in that report to update each flow receiver information of the network. The second type of QoS update is a reply to the signaling commands from a flow receiver which is to be introduced in Section 6.4.3. In such an answer message, a network component will similarly report its current QoS status for the information of the flow receiver. The last QoS update is a routine information sharing process as is defined in the push mode in Section 4.5.1.

In essence, each service status table records the “most recent” information of a QoS component that is known to the flow receiver and QMan middleware. It is used as historical data in the evaluation algorithm described below for the selection of a suitable network QoS component for end-to-end adaptation. It should be noted that runtime QoS status of a network QoS component is always changing, depending on factors such as equipment conditions, routing topology and traffic load. Hence QoS information regarding a network QoS component, as maintained by QMan, may not reflect the “exact current” QoS status of a QoS component. In this sense, the adaptation evaluation performed by QMan is actually upon the “last known” information of all QoS components. Of course, more up-to-date information about a network component can be learned if the signaling frequency between end-hosts and networks is to be increased. However, frequent information exchange may pose severe overhead on network performance. In a real end-to-end QoS system which requires accurate information sharing of network QoS components status, the information sharing frequency can be tuned to balance between the signaling overhead and the information accuracy.

### 6.4.2 Cross-component Adaptation Evaluation Algorithm

In QCMF, a QoS violation is identified by a flow receiver in two ways: observation of flow abnormal through end-to-end monitoring or receipt of a violation report from a network QoS component. The latter is used for the receiver to double-confirm the appearance of a QoS violation and to locate the physical place of the violation. In either case, the evaluation unit of QMan will select a suitable one, among all end-to-end (network and end-host) QoS components for adaptation. In view of the need for a fast recovery from violations, QCMF defines the following heuristics for end-to-end adaptation: (1) system-level end-host adaptations (e.g., swapping media codec) is set as a default choice which is typically able to bring the end-to-end QoS back to a normal state most quickly, (2) application-level QoS degradation (e.g., reduce video frame rate) is defined as the last adaptation choice since it will inevitably affect the perceptual quality of an end-user, (3) cross-component adaptation evaluation will be performed to find a network adaptation solution that can sustain transmission quality if (1) is not achievable. Such heuristics can be modified by end-users so that different adaptation sequences can be realized in respective end-to-end environments.

The cross-component adaptation evaluation algorithm basically considers two factors in the selection of a network QoS adaptation: (1) the *adaptation cost* at a network QoS component, and (2) the *possibility* that an issued adaptation request from QMan would be rejected by a network QoS component. We will address these two points one by one. The skeleton of the evaluation algorithm is listed in Figure 6-3.

```
Step 1): Review the service status table of a network QoS component.
Step 2): Determine the necessary inter-service adaptation (i.e., service class alteration) in that component for the problem solving of the QoS violation; caculate corresponding adaptation cost.
Step 3): Compute the possibility $p_{reject}$ that the issued adaptation request would be rejected by the QoS component based on its service model and traffic load information.
Step 4): Caculate the utility factor of the adaptation at the QoS component which is defined as $\frac{1}{\cos t * p_{reject}}$.
Step 5): Repeat steps 1-4 for all network QoS components.
Step 6): Select a QoS component of largest utility factor for adaptation. If multiple QoS domains have the same utility, select the one with lowest $p_{reject}$.
```

Figure 6-3: Skeleton of the cross-component adaptation evaluation algorithm

*A. Definition of adaptation cost*

For each network QoS component, the adaptation cost generally refers to a particular concern with respect to the additional resource expenditure incurred by that adaptation. For example, in mobile terminals where power consumption is a primary concern, the cost can be defined as the difference between power usage before and after adaptation. In a QoS-enabled network, such cost may be defined as the money paid for service upgrade/degrade. Suppose that service class 2 of network QoS component $i$ undergoes a traffic overload and hence an end-to-end flow which subscribes to this service class encounters a delay violation. As explained, such a delay violation can be solved within the source network QoS component $i$ by service upgrade, or be compensated by QoS adaptation from another end-to-end network QoS component $j$ that participates in the end-to-end collaboration. However, the cost for adaptation in different network QoS component can be different. The reason is simply that, for example, the unit bandwidth allocation price at component $j$ is higher than that at component $i$. In view of the comparable QoS adaptation conditions, our evaluation algorithm will try to select a adaptation choice with lowest adaptation cost.

### *B. Definition of* $p_{reject}$

On the other hand, a QoS adaptation request issued by receiver's QMan middleware risks being rejected by a network QoS component, if the requested adaptation is not achievable. Recall that the receiver host evaluates the QoS status of each network QoS component based on its "latest known information" (Section 6.4.1), which is refreshed each time the receiver communicates with that network component. However, such information may not precisely reflect the current load information of that network component since its traffic conditions is keep changing. For example, a network component may have just accommodated several new flows from other end-hosts which make its service classes almost full. In the case that such information has not been promptly updated with the receiver of the end-to-end flow in concern, a service upgrade request from QMan would be rejected by the network due to insufficient bandwidth. To conclude, such a situation arises due to the synchronization problem between a flow receiver and a network QoS component. Once an inappropriate adaptation decision is made, time would be wasted in such request/deny signaling process between a flow receiver and a network component. As a result, user perception would be interrupted due to continuous transmission quality variation.

A straightforward solution out of this puzzle is to let a flow receiver grasp the most recent QoS information of all network QoS components each time an adaptation decision is to be made. However, we find that there are a few intricate problems associated with such an idea. Firstly, it may take a quite long time for QMan to collect the precise information of all network QoS components. For instance, it will take round trip time to receive status update from each QoS component even if we omit all other possible processing delay and assume that propagation delay for all network QoS components is the same. Secondly, there is no guarantee that the receiver will obtain the precise QoS information of a network component at the moment of QoS evaluation even if we frequently exchange information. This is simply because the workload status of a QoS component is always changing. It is possible that the working status of a network QoS component is altered (1) just after its recent update with a flow receiver, or (2) just before a flow receiver finishes evaluation and issues an adaptation request. In either case, a wrong decision would be made at the flow receiver based on the incorrect end-to-end information. Lastly, frequent message exchange can provide more accurate information of a network QoS component, however, at the cost of imposing a burden on the network and may not be applicable in practice due to scalability concern. As can be concluded, there is no approach that can strictly guarantee the “freshness” of information about a network QoS component from a flow receiver’s point of view due to the distributed and long physical distance nature of network transmission.

To smooth the impact of the information synchronization problem and to best avoid the case that an adaptation request would be rejected by a network component, we take an end-to-end approach to estimate the workload status of a network component. We introduce a parameter $p_{reject}$ to denote the possibility that an adaptation request would be rejected by a network QoS component. Let $T$ be the throughput of an end-to-end flow that requests for an service upgrade adaptation at QoS component $m$, $C$ be the bandwidth capacity of the target service class $S_{m,b}$ (i.e., the service class to upgrade to), and $(R_i, T_i)$ be the $i^{th}$ historical record of the load information of component $m$ known to the receiver (see Table 6-2), $p_{reject}$ can be expressed as:

$$p_{reject} = \Pr\{T + R_{current} > C \mid \{(R_1, T_1), (R_2, T_2) ... (R_i, T_i)\}\}, i \in (1, n) \,, \qquad (1)$$

where $R_{current}$ is the (actual) current work load of service class $S_{m,b}$ at network QoS component $m$.

It can be seen that once the receiver knows the actual load $R_{current}$ at service class $S_{m,b}$, it can judge whether a service upgrade is feasible or not. However, $R_{current}$ is always not available to the receiver due to synchronization problem as discussed. Hence, we will try to calculate the statistical value of $R_{current}$ based on information such as the service model of $S_{m,b}$ and the historical data known to the receiver. Once $R_{current}$ is estimated, $p_{reject}$ can be deterministically derived from formula (1) (i.e., the result is either 0, meaning that a request will be accepted or 1, meaning rejection). It should be noted that as $R_{current}$ is estimated statistically, the value of $p_{reject}$ we get is actually of statistical meaning which reflect the most possible situation whether an adaptation request would be accepted or not.

A service class $S_{m,b}$ can be viewed as a queuing system holding limited permits and thus can accommodate limited number of customers simultaneously. A couple of traffic models have been developed to model the service status of a priority queue [140] and its possible QoS characteristics [141]. These models are intended for admission control at a QoS-enabled network to check whether there are sufficient resources (e.g., bandwidth) to entertain an incoming service request or not. Given parameters such as customer arrival rate $\lambda$ and service rate $\tau$, these models can statistically compute the current queue status which in the general form is:

$R_{current} = f_{\mathrm{mod}el}(\lambda, \tau)$, where $\mathrm{mod}el$ is the established service model of service class $S_{m,b}$.

Typically, parameters $\lambda$ and $\tau$ are pre-defined and tuned through runtime traffic monitoring and model validation process in respective systems [142]. Making use of these researches, we are able to obtain $R_{current}$ for each QoS component once its traffic model has been setup. The accuracy of $R_{current}$ is a reflection of how well the established service model has reflected the traffic statistics of the network component and is the research focus in many relevant literatures [38][143][144].

## C. A simple $M/M/1/M$ model for the estimation of $p_{reject}$

In the following sections, we consider a simplified traffic model to calculate $p_{reject}$ for illustration purpose. Let us assume that all customers subscribing to a service class demands for unit amount of bandwidth. Hence a service class can be viewed as a leaky bucket system with limited amount of tokens (equals to its bandwidth capacity $C$). We further assume that the customer arrival rate in that service class conforms to Poisson distribution and service rate conforms to exponential distribution. Such a service class can be modeled as a $M/M/1/M$ queue [145]. Similar to the classic queuing analysis for $M/M/1/M$ system, we have $p_i\lambda = p_{i+1}\mu$, where $\lambda$ and $\mu$ are the arrival rate and service rate of the service class respectively and $p_i$ is the probability that $i$ flows stay concurrently in the service class.

Given $\sum_{n=1}^{n=C} p_n = 1$ and $\rho = \lambda/\mu$,

we obtain $\sum_{n=1}^{n=C} \rho^n p_0 = 1$, then we get $p_0 = \frac{\rho(1-\rho^C)}{1-\rho}$, and $p_{reject} = p_n = \frac{\rho(1-\rho^C)}{1-\rho}\rho^C$

Note that due to the memoryless character of the exponential distribution of the service rate $\mu$, $p_{reject}$ is independent of the historical data in this $M/M/1/M$ model. Hence there is no need for the receiver to record the historical information for a QoS component. In a more practical queuing system as modeled in [140][141], historical information is essential to calculate $p_{reject}$ since assumptions such as passion arrival rate is broken. Moreover, the load status of a network QoS component is changing from time to time which also alters the traffic characteristics of that network. Hence configuration parameters of the traffic model such as $\lambda$ and $\tau$ are often tuned from time to time in relevant traffic engineering researches for a most precise description of network traffic status. In the context of end-to-end QoS management, QCMF requires these parameters to be updated with the flow receiver once their values have been adjusted in a network QoS component.

### *D. End-to-end adaptation evaluation criteria*

As discussed, QCMF considers both adaptation cost and $p_{reject}$ in selection of an end-to-end adaptation solution. In principle, a network QoS component of lower adaptation cost has a better chance of being selected for adaptation. On the other hand, a QoS component of heavy traffic load will present a large $p_{reject}$ and in consequence, would unlikely be selected so as to avoid a possible adaptation rejection and failure. In view of both considerations, the final evaluation is based on the *utility factor* of each QoS component, which is defined as $\frac{1}{\cos t * p_{reject}}$. A utility factor seeks the balance between the reward (gain of cost) and penalty (chance of been rejected) of each possible adaptation choice. In QCMF, a network QoS component of largest utility factor will be finally selected for end-to-end QoS adaptation.

### 6.4.3 End-to-end Signaling and Adaptation Algorithm

In addition to information sharing and end-to-end adaptation, QCMF also defines the following primitives for the signaling and coordination actions among end-hosts and network QoS components in the control plane.

- Notification

The notification primitive is used by end-hosts and network QoS components to inform each other of QoS transactions. Firstly, it is used by a network QoS component to inform the flow receiver the appearance of a QoS violation in its local region. Information to be reported includes the initial agreed QoS level and current QoS status (in the form of parameters such as delay, jitter and traffic model configurations such as $\lambda$ and $\tau$). Secondly, the reception of such a message at the receiver end-host will trigger an end-to-end adaptation evaluation as discussed. A QoS component of best utility factor will be selected for QoS adaptation, which subsequently will receive a notification message from the flow receiver for the adaptation details (e.g., the service class to upgrade to).

- ACK & Reject

Upon receiving an adaptation request, a network QoS component should check its service status to see whether the required service upgrade adaptation is achievable or not. If the admission test is passed, the QoS component will execute the adaptation immediately, after which it will reply the receiver with an ACK message, informing the latter the success of QoS adaptation. The updated information of this QoS component will also be piggyback in the ACK message. Otherwise if an adaptation is not applicable, the QoS component will reply the flow receiver with a Reject message denying the adaptation. Once the receiver receives such a message, it will (1) select the network QoS component of second best utility factor for adaptation notification if there is no QoS status update from each network component since last decision-making, or (2) trigger another round of adaptation to calculate utility factor if there is any new QoS update.

- Request

This primitive is used by the receiver to request for the updated information of a network QoS component. As discussed, excess message exchange between the flow receiver and a network QoS component can pose a heavy burden on the network and is also not scalable in large networks. Hence we restrict the usage of the Request message to the scenario where the information of a QoS component has not been updated for a period longer than a pre-specified timer. The specification of such a timer can be designated and tuned through an engineering process in an actual end-to-end system.

## 6.5 Simulation Results

| **Network Component** | **Service class** | **Unit price** | **Bandwidth requirement** | **Cost of adaptation** | **Rejection probability** | **Utility factor** |
|---|---|---|---|---|---|---|
| QoS Component 1 | 2 | $4/h/Mbps | 2Mbps | 8 | 0.25 | 0.5 |
| QoS Component 2 | 2 | $1h/Mbps | 2Mbps | - | - | - |
| QoS Component 3 | 3 | $7h/Mbps | 2Mbps | 14 | 0.20 | 0.36 |
| QoS Component 4 | 3 | $2/h/Mbps | 2Mbps | 4 | 0.20 | 1.25 |
| QoS Component 5 | 2 | $3/h/Mbps | 2Mbps | 6 | 0.15 | 1.11 |

Table 6-3: Service subscription settings of a flow in simulation and the its utility factor

We validate our end-to-end QoS adaptation scheme through simulations under NS-2 environment [146]. The end-to-end QoS provisioning path is composed of 5 adaptive network QoS components in a chain topology (Figure 6-1). Key configurations of the experiments are as follows. In each QoS component, we define 4 service classes of limited bandwidth, each of which produces different QoS services in terms of delay guarantees. Each network QoS component has a service options table defined similar to the one shown in Table 6-1. The QoS information of each network component is shared with end-hosts at session start-up time and is updated at runtime. Hence, the flow receiver maintains a service status table similar to Table 6-2 for each network QoS component. We use a $M/M/1/M$ queuing system to model network QoS components where each of them is modeled with different values of $\lambda$ and $\mu$. The propagation delay between QoS components and the flow receiver is set to around 1ms. The processing delay at each network QoS component is set to an average value of 7ms with normal distribution.

The simulation is executed as follows. A media flow is delivered from the sender to the receiver and subscribes to a certain service class in each network QoS component. At time 12s (0.2 minute in Figure 6-5), a delay violation happens at network QoS component 2 which is subsequently detected by that component and is subject to rectification. Such a delay violation is simulated by sending a network traffic flow to the service class that the end-to-end multimedia media flow is entertained. Such a traffic flow has overloaded the service class and hence the media flow suffers from a delay violation. Upon appearance of such a violation, a violation report is sent by the network QoS component to the flow receiver which triggers the end-to-end evaluation process to calculate the utility factors of all end-to-end QoS components (Table 6-3). After that, network QoS component 4 (which is of the largest utility factor) is selected for end-to-end adaptation. An adaptation notification is subsequently sent to QoS component 4 where a service upgrade adaptation is conducted. As a result, the overall end-to-end delay restores to the acceptable range (at time around 0.25 minute in Figure

6-5). Half minute later, the traffic flow is removed and hence the traffic jam disappears at QoS component 2 which restores it local QoS status to a normal situation. Another QoS status update is sent from component 2 to the flow receiver. By comparing the service status tables of both network QoS component 2 and 4, the flow receiver finds that current end-to-end subscription cost (the sum of individual subscription cost at each network QoS component) is higher than the one before adaptation. This is because the unit bandwidth allocation price in current service class of component 4 is higher than that in component 2. Hence, the flow receiver notifies both QoS component 4 to downgrade its subscription to the original service class.

The simulation results are shown in Figure 6-4 and Figure 6-5 respectively. The delay violation in QoS component 2 (at time 0.2 minute) is compensated by a service level upgrade in QoS component 4 (at time 0.22 minute) under the guidance of the flow receiver. This time interval is the overhead for the distributed coordination among end-hosts and networks as well as the utility factor computation and adaptation evaluation within the flow receiver. Similar observation can be found in the later "roll-back" process (at time 0.6 minute) where the traffic flow is removed from network QoS component 2. During the process, the end-to-end delay remains relatively stable as is shown in Figure 6-5, except two glitches (at time 0.2 minute and 0.6 minute respectively) which reflect the end-to-end QoS fluctuation during violation and adaptation. Note that the second stage of adaptation is an optional step in current implementation of QCMF. Such an action is a coordinated process which reviews the new QoS status of component 2 and 4 and the overall end-to-end subscription cost. The benefit of such an additional adaptation is that a more optimal global QoS can be achieved, however at the cost of additional information signaling and adaptation scheduling.

Lastly, we evaluate the performance of our adaptation evaluation algorithm (named as AEA for simplicity). We compare it with another aforementioned generic information synchronization scheme: no adaptation decision will be made at the flow receiver until it synchronize and get the latest QoS information update from all network QoS components (denoted as Adaptation after Status Update - ASU). The configuration parameter settings are identical to those described in the previous experiment. Both AEA and ASU use the same evaluation algorithms and signaling primitives described in Section 6.4. As has been explained, an adaptation request issued by a flow receiver in both AEA and ASU algorithms risks being rejected by a network QoS component due to insufficient resources for service upgrade. However, it takes different time for these two algorithms to finish a QoS adaptation process under such a rejection situation. Let us denote the propagation delay among network QoS components and the end-host as $T_p$, the evaluation delay at the flow receiver as $T_e$, it takes $4T_p + T_e$ time for ASU to work out an adaptation decision, where the four propagation delay is consumed by the receiver to (1) receive a violation report from a network QoS component, (2) request for information update from all network QoS components, (3) receive QoS update of all components based on which end-host evaluation is carried out, and finally (4) inform the selected QoS component for adaptation. As said, we ignore all other possible processing overhead. If an issued adaptation request as in step (4) is rejected by the target network QoS component, it takes another $4T_p + T_e$ time for the receiver to finish the second round of information gathering, adaptation evaluation and event notification. Comparatively, AEA takes only $2T_p + T_e$ to evaluate and notify an adaptation, where $2T_p$ is the delay to receive a violation report from the network component in trouble and issue an adaptation request. If the adaptation request is denied by a network QoS component, it takes another $2T_p + T_e$ time for the flow receiver to execute the second round of adaptation. The performance evaluation of two algorithms under the simulation setting described before is shown in Figure 6-6. It can be concluded that, if a proper traffic model is selected to describe the traffic characteristics of a network QoS component, our AEA algorithm can achieve higher adaptation efficiency in terms of both adaptation latency and signaling overhead.

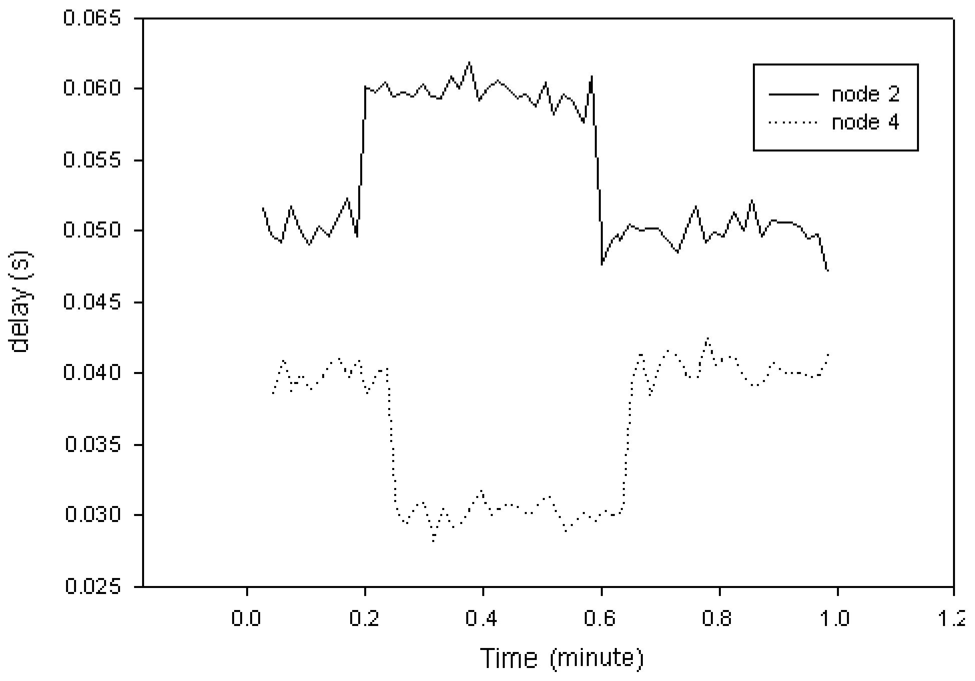


Figure 6-4: Delay change at network QoS component 2 where a violation happens and component 4 which participates in the end-to-end collaboration to solve the violation

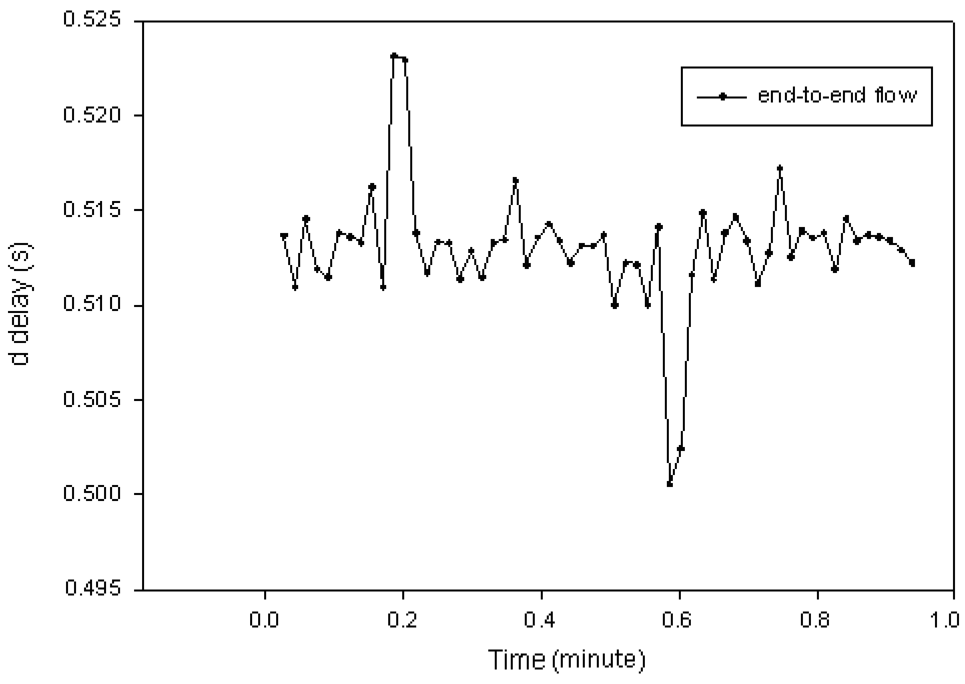


Figure 6-5: Experienced end-to-end delay before/after a delay violation

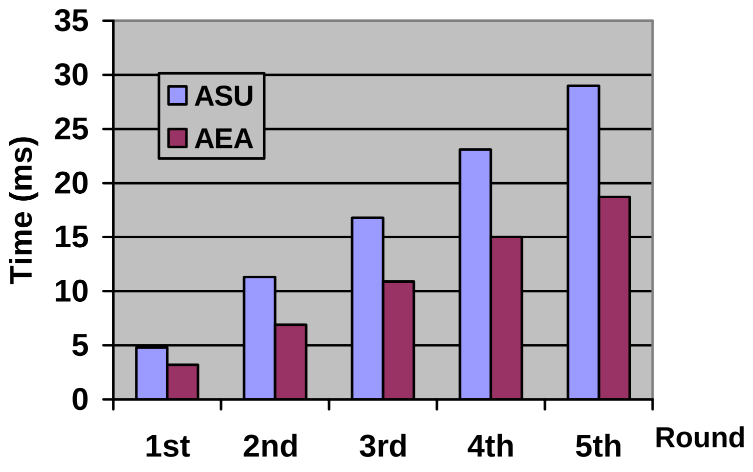


Figure 6-6: Delay overhead of adaptation algorithms AEA and ASU for message exchange and signaling among network QoS Components and end-hosts

## 6.6 SUMMARY

In this chapter, we propose a cross-component QoS coordination scheme for end-to-end QoS adaptation. In this scheme, QCMF models all QoS services along the provisioning path and provides mechanisms for QoS coordination and adaptation upon the occurrence of a QoS violation. Residing in end-hosts, QMan middleware runtime copes with a QoS violation by selecting a suitable QoS adaptation solution among all the end-to-end candidate choices. Such a selection focuses on the utility factor of each possible choice and is directed by end-to-end heuristics and policies. Analysis and preliminary simulation results validate the design and feasibility of our approach for end-to-end QoS coordination.

# CHAPTER 7
# IMPLEMENTATION AND EVALUATIONS

In this chapter, we present the implementation of our QoS control and management framework. The objective of this prototype is to demonstrate the working principle and management performance of QCMF. As discussed earlier, the management functions of QCMF is mainly composed of two parts: (1) QoS negotiation and configuration at build-time, and (2) QoS monitoring and coordinated end-to-end adaptation at runtime. We will first describe the experiment scenario and testbed configurations. After that, performance analysis of various management functions in QCMF will be introduced respectively.

## 7.1 IMPLEMENTATION SCENARIO

We have setup a testbed for experiments as shown in Figure 7-1. Two groups of PCs are separated by an IP router in between and hence form two subnets. The configuration of each computer is shown in Table 7-1. We deliver audio and video flows from the stream sender to the stream receiver. The audio source outputs an audio flow of PCM format, 44100Hz and stereo quality. It is transcoded into either G723_RTP or MPA_RTP format for media streaming over the network, depending on the application QoS requirements and runtime conditions. The video source outputs a video flow of format MPEG-1, 640*256 resolution at constant rate of 30fps. It is transcoded into either MPEG_RTP or JPEG_RTP format for transmission. We write the A/V transmission application on J2SE 6 platform using JMF library [58]. The RTP transmission between the stream sender and receiver uses UDP as the transport layer protocol.

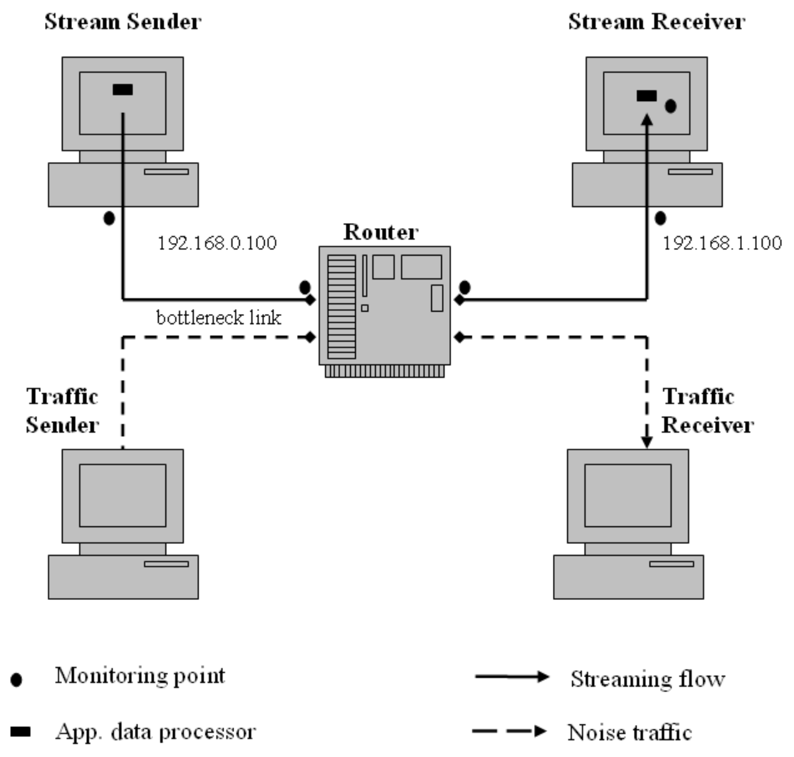


Figure 7-1: Testbed environments

The Linux router in between is capable of adjusting bandwidth allocation a specific queue (RED queue or FIFO queue), and hence provide a generic form of service differentiation of network traffic. The stream sender, receiver and the router constitute the complete end-to-end path where our QCMF serves on. A traffic generator software hosted in the traffic sender is engaged to produce huge amount of data traffic to the traffic receiver sitting in the other subnet. Such background traffic will influence the bandwidth allocation of the audio/video streaming we are testing so that network congestion will appear. A software monitor capitalizing on WinPcap library [147] is placed at the streaming receiver to capture incoming traffic on the NIC and calculate end-to-end traffic statistics. On detecting a QoS violation, QCMF will perform end-to-end adaptation according to pre-defined heuristics, policies and evaluation algorithms as discussed in Section 6.4.2. The end-host/middleware QoS adaptation is directed by QMan and enforced by the DPF. Currently, QCMF supports 11 media protocols, namely RTP, TCP, UDP, IP, G711 codec, G723 codec, GSM codec, MPEG codec, H263 codec, JPEG codec and affined encryption/decryption at the middleware level. We also integrate a frame rate monitor with the multimedia transmission application which measures the velocity of video display. As explained, video frame rate is defined as the example application QoS violation indicator with the violation delimiter set to 25fps.

| Computers | Configurations |
|---|---|
| Stream sender | PC, Core 2 processor 2.13GHz, 2GB RAM, one Gigabit network card, Windows XP SP3 |
| Stream receiver | PC, Pentium 4 processor 2GHz, 1GB RAM, one Gigabit network card, Windows XP SP3 |
| IP Router | PC, Pentium 3, 933 MHz CPU, 512MB RAM PC, two 10/100 Mbps NIC, Fedroa 9 |
| Traffic generator | PC, 10/100 Mbps NIC, Windows XP SP3 |
| Traffic receiver | PC, 10/100 Mbps NIC, Windows XP SP3 |

Table 7-1: Testbed configurations

## 7.2 QoS Knowledge Processing

We have first established a knowledge plane for information modeling and exchange using the aforementioned techniques. All QoS knowledge is modeled by ontology in RFDS language; knowledge structures of multimedia applications, QMan/DPF middleware and generic QoS-enabled network are established respectively. The overall knowledge ontology consists of 91 ontology classes and 224 properties. The Jena semantic web toolkit [148] has been chosen as the tool to conduct QoS knowledge reasoning. We have evaluated our knowledge plane infrastructure in terms of the performance of ontology modeling, knowledge reasoning and knowledge sharing. The experiment results are elaborated in the following sections, which in general demonstrate the feasibility of our approach for real-time QoS knowledge management.

## 7.2.1 SQS Initiation Delay

In the first step, we have conducted experiments to evaluate the overhead of the two-layer application QoS ontology model of SQS. As explained in Section 4.3, different QoS middleware may define QoS classification trees of different content and depth, depending on the capability of the QoS middleware and runtime environments. In this test, QoS middleware at the media sender and receiver is designed of different abilities: the media receiver is a light-weighted laptop that defines 9 QoS base-classes and derives 4 QoS domains from them; the media sender (server) is more powerful PC which defines 9 QoS domains that merge from 13 QoS base-classes. The operation of merging QoS base-classes into QoS domains is guided by configuration rules as explained before. The latencies for loading of QoS base-classes and merging of QoS domains have been recorded and compared in Figure 7-2. It is not surprising to see that the delay overhead of SQS is proportional to the size of dataset (i.e., number of QoS base-classes and QoS domains). However, such initiation delay is not significant (less than 0.6s) at both communication parties.

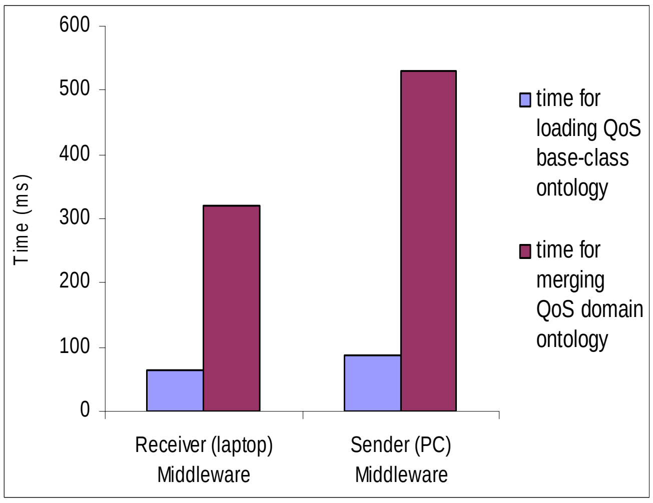


Figure 7-2: Overhead of the two-layer ontology design

### 7.2.2 Knowledge Reasoning Performance

We have next evaluated the performance of QoS knowledge reasoning over different scales of dataset. As mentioned in Section 4.5.2, knowledge reasoning in QCMF consists of two parts: ontology reasoning and user-defined reasoning. We first measure the performance of ontology reasoning which is guided by reasoning rules built-in RDF/RDFS standard. The size of dataset is measured in terms of the number of RDF triples, each of which represents a single statement (S-V-O predicate). Technically, these triples are generated by the middleware knowledge reasoner through parsing and converting all RDF class definitions contained in the knowledge base. Ontology reasoning is enforced to derive implied knowledge once all RDFS definitions have been compiled into triples at build-time. In addition to offline QoS analysis, ontology reasoning may also be executed runtime upon insertion of new QoS knowledge through knowledge sharing process. However, such reasoning will only cover partial (i.e., those relevant) knowledge for checking and hence is much less time-consuming than build-time reasoning.

Figure 7-3 shows the results of ontology reasoning on PCs and laptops of different CPU speeds, which demonstrate that the ontology reasoning latency of RDF is proportional to the size of dataset. However, as increments are approximately linear in nature, the delay is still acceptable even for a large dataset of 2261 triples (roughly correspond to 302 RDFS classes, which are sufficiently large enough for modeling more than, for example, 100 protocols). The ontology reasoning latency also depends on the CPU speed of an end-host. A PC with higher CPU clock speed will require less time to prepare the knowledge reasoner. However, the difference in performance is not significant as is shown in Figure 7-3.

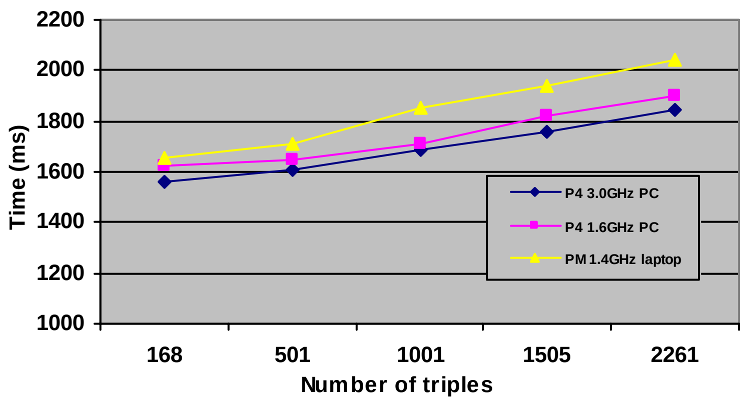


Figure 7-3: The ontology reasoning performance

User-defined reasoning is build-time/runtime performed based on user-defined rules. The main task here is to derive QoS knowledge for the preparation of subsequent QoS activities (e.g., QoS violation analysis or adaptation coordination). We have measured the performance of user-defined reasoning and compared it with that of ontology reasoning. The experiment is also conducted with respect to the different scales of QoS knowledge as shown in Figure 7-4. The experiment result shows that the time needed for ontology reasoning is larger than that for user-defined reasoning. This is probably due to a large set of rules built-in standard ontology definition and reasoning whereas only 14 rules are defined for user-defined reasoning in QCMF currently (some of which has been explained in Section 4.5.2). We also notice that latency of ontology reasoning and user-defined reasoning will becomes unbearable for real-time processing if the data size grows significantly large (e.g., more than 2000 ontology classes and properties – artificially replicated here for testing purpose only). Although we believe most end-to-end QoS ontology would be sufficiently modeled in hundreds of ontology definitions based on our experiences (e.g., our end-to-end QoS management prototyping contains a few more than 100 ontology definitions), further research is needed to improve the performance of knowledge reasoning in an resource-scare environment (e.g., smart phones with wireless networking).

After the above two knowledge reasoning steps, we found that the knowledge reasoner can answer a query within tens of milliseconds (i.e., processing delay) in both small and large scale dataset settings. Such a query may be issued within a QoS component (e.g., protocol retrieval) or from an external source (e.g., knowledge query from other QoS components in the pull mode). Meanwhile, the memory consumption for knowledge reasoning is around 7MB in both small and large data settings. In general, all these experiment results suggest that it is feasible to employ an ontology-based approach to model and reason end-to-end QoS knowledge even on less powerful hosts such as laptops.

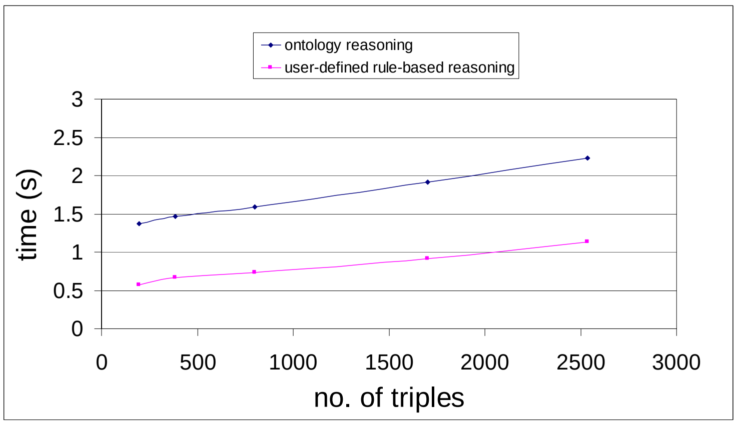


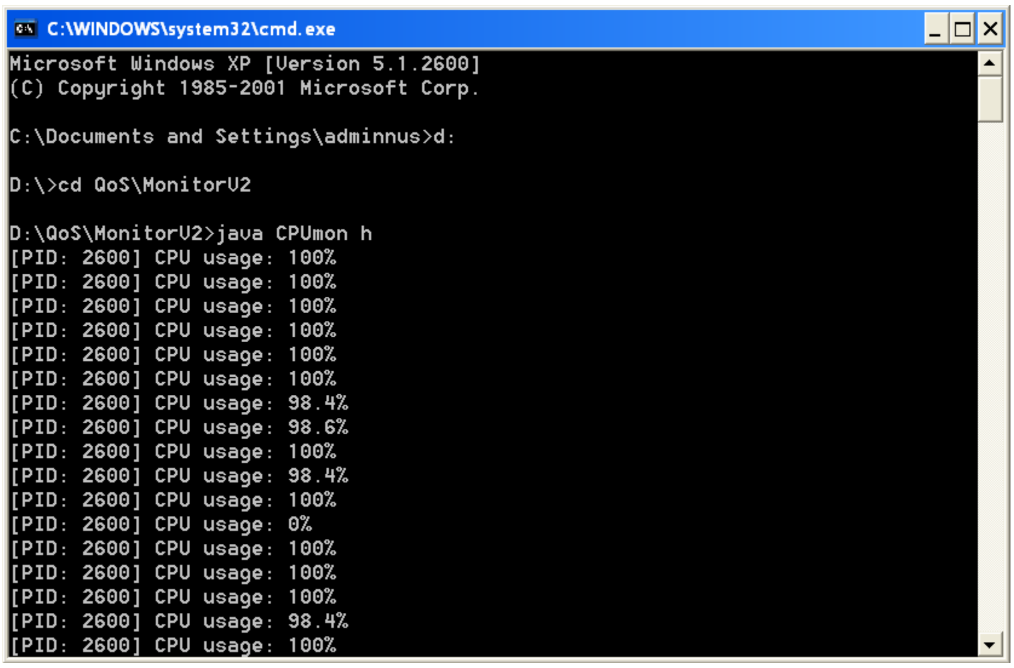
Figure 7-4: Knowledge reasoning performance comparison

## 7.3 QoS Violation Analysis

We have subsequently conducted experiments for QoS violation analysis in various environments according to the violation types defined in Table 7-2. We first validate our ideas and approaches in a controlled testbed environment. We manually create different kinds of QoS violations and classify them using our end-to-end violation analysis scheme introduced in Chapter 5. After that, we test our systems on an open campus network where network QoS violations are observed in both wired and wireless environments. Lastly, we repeat the experiments in a real Internet environment - PlanetLab platform [149] and collect violation data accordingly. Analysis of the collected data in the above environments shows the validity and effectiveness of our system in terms of both training/classification accuracy and overhead.

| **Violation types** | **Possible violation nature** | **Available adaptation solutions** |
|---|---|---|
| Sender host violation | Sender machine busy | Switch codec, change streaming server, reduce QoS |
| Receiver host violation | Receiver resource scarcity | CPU/buffer re-allocation, switch codec, reduce QoS |
| Network error | Hardware failure, link error | Change route, re-transmission, upgrade service class |
| Network traffic violation | Network congestion | End-to-end congestion control, media compression |

Table 7-2: QoS violation classification in view of controllable resources and available end-to-end adaptation choices



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\adminnus>d:

D:\>cd QoS\MonitorV2

D:\QoS\MonitorV2>java CPUmon h
[PID: 2600] CPU usage: 100%
[PID: 2600] CPU usage: 100%
[PID: 2600] CPU usage: 100%
[PID: 2600] CPU usage: 100%
[PID: 2600] CPU usage: 100%
[PID: 2600] CPU usage: 100%
[PID: 2600] CPU usage: 98.4%
[PID: 2600] CPU usage: 98.6%
[PID: 2600] CPU usage: 100%
[PID: 2600] CPU usage: 98.4%
[PID: 2600] CPU usage: 100%
[PID: 2600] CPU usage: 0%
[PID: 2600] CPU usage: 100%
[PID: 2600] CPU usage: 100%
[PID: 2600] CPU usage: 100%
[PID: 2600] CPU usage: 98.4%
[PID: 2600] CPU usage: 100%
```

Figure 7-5: CPU occupier program for CPU violation at end-hosts

### 7.3.1 Testing Cases

We have first conducted a CPU violation test on the testbed platform and collected corresponding violation data. A utility program is designed (Figure 7-5) which can occupy CPU time slices at several scales, e.g., minimal (occupy 10%-20% CPU time), medium (around 50%) or maximum (80%-90%). By tuning the “volume” of the CPU occupier program at either media sender or the receiver, a CPU violation can be observed in the media streaming. Figure 7-6 shows the values of the application QoS violation indicator (i.e., frame rate) and flow descriptors during a 120-second CPU contention test. In the first 60 seconds, no external interference is injected and hence the video streaming presents satisfactory performance. In the next 60 seconds, the CPU occupier program is launched to contend with the streaming application for CPU time slice. A few QoS violations can thus be observed where frame rate drops below 25fps (Figure 7-6(d)). Flow level statistics such as packet delay and jitter are recorded in Figure 7-6(a) and Figure 7-6(b) respectively which shows their variations before and during the violation. However, no packet loss is observed in this CPU violation test as can be seen from Figure 7-6(c).

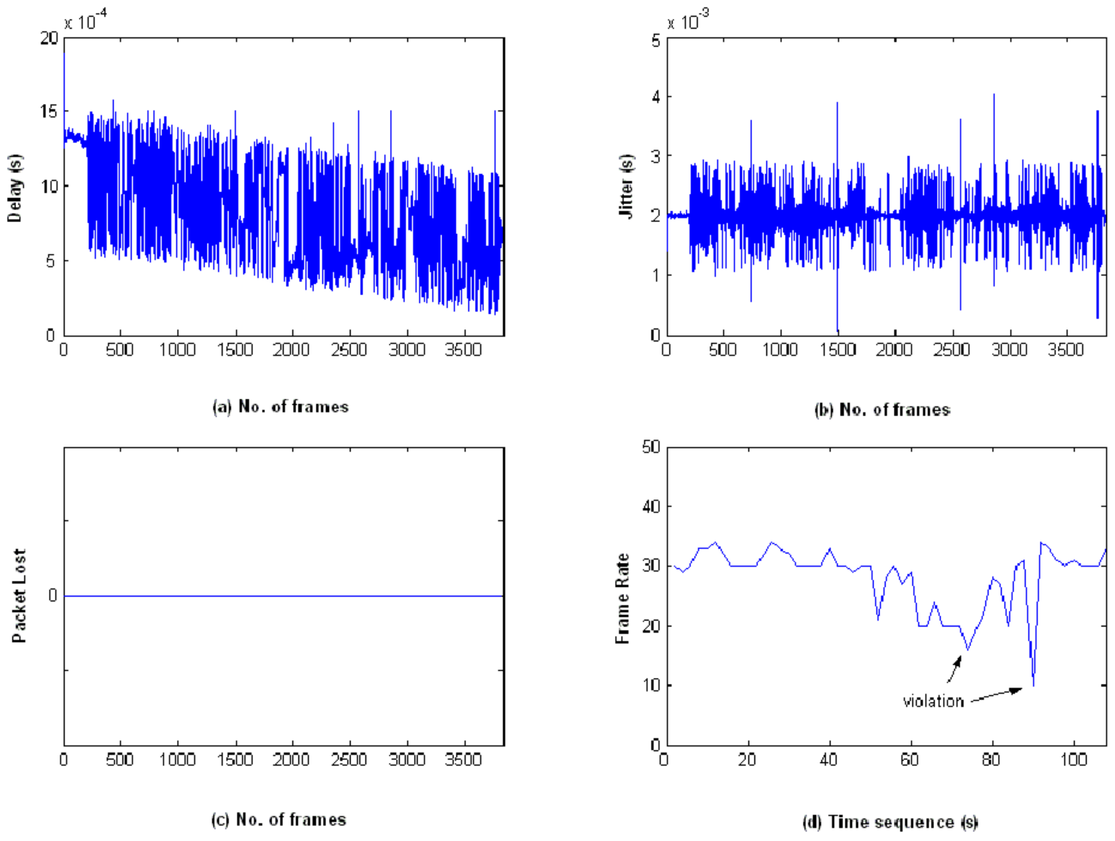
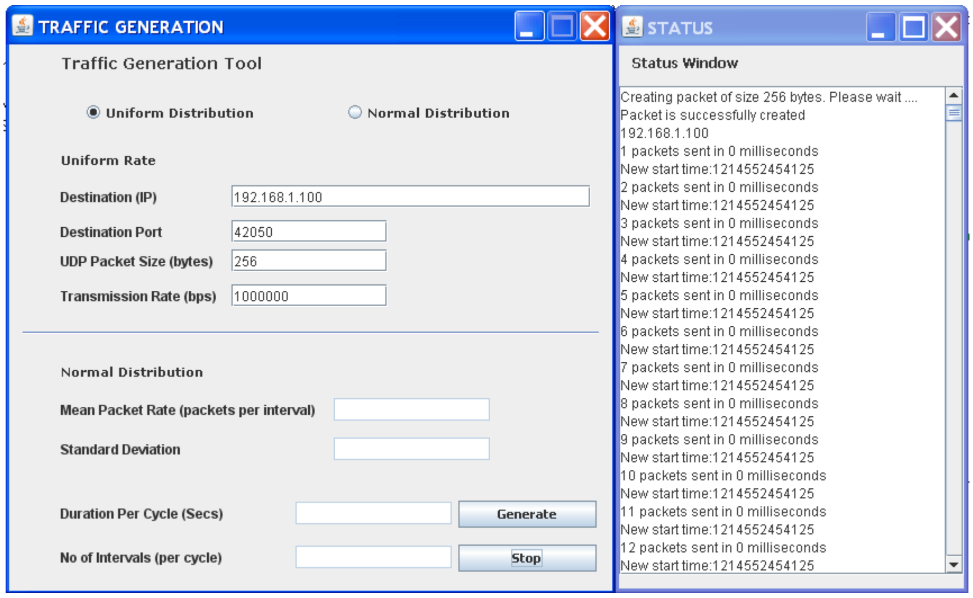


Figure 7-6: Observation of end-to-end QoS w/ and w/o CPU contention

Secondly, we have designed a network congestion test. A traffic generator is programmed (Figure 7-7) which can generate traffic of either constant rate (emulate Internet background traffic such as constant rate FTP flows) or normal distribution (emulate the arriving traffic experienced by a core Internet router). The traffic generator is launched during streaming which deliver packets to the traffic receiver sitting in the other subnet. By outputting a large volume of data, the traffic generator creates a congestion link on the end-to-end path of the video transmission. Figure 7-8 shows the measured performance of the video streaming in another 120-second test. In the first 60 seconds, the traffic generator program yields a large volume of traffic of normal distribution which leads to the performance degradation of the video flow. Packet loss is observed during this period which indicates overflow of the router queue (Figure 7-8(c)). The traffic generator is removed in the next 60-second testing. As can be seen from the last part of Figure 7-8(d), quality of the video flow restores gradually, where frame rate varies within the acceptable range.


TRAFFIC GENERATION

Traffic Generation Tool

(● Uniform Distribution) ○ Normal Distribution

Uniform Rate

| | |
|---|---|
| Destination (IP) | 192.168.1.100 |
| Destination Port | 42050 |
| UDP Packet Size (bytes) | 256 |
| Transmission Rate (bps) | 1000000 |

Normal Distribution

| | |
|---|---|
| Mean Packet Rate (packets per interval) | |
| Standard Deviation | |
| Duration Per Cycle (Secs) | |
| No of Intervals (per cycle) | |

STATUS

Status Window

```
Creating packet of size 256 bytes. Please wait ....
Packet is successfully created
192.168.1.100
1 packets sent in 0 milliseconds
New start time:1214552454125
2 packets sent in 0 milliseconds
New start time:1214552454125
3 packets sent in 0 milliseconds
New start time:1214552454125
4 packets sent in 0 milliseconds
New start time:1214552454125
5 packets sent in 0 milliseconds
New start time:1214552454125
6 packets sent in 0 milliseconds
New start time:1214552454125
7 packets sent in 0 milliseconds
New start time:1214552454125
8 packets sent in 0 milliseconds
New start time:1214552454125
9 packets sent in 0 milliseconds
New start time:1214552454125
10 packets sent in 0 milliseconds
New start time:1214552454125
11 packets sent in 0 milliseconds
New start time:1214552454125
12 packets sent in 0 milliseconds
New start time:1214552454125
```

Figure 7-7: Traffic generator can produce traffic of either constant rate or normal distribution

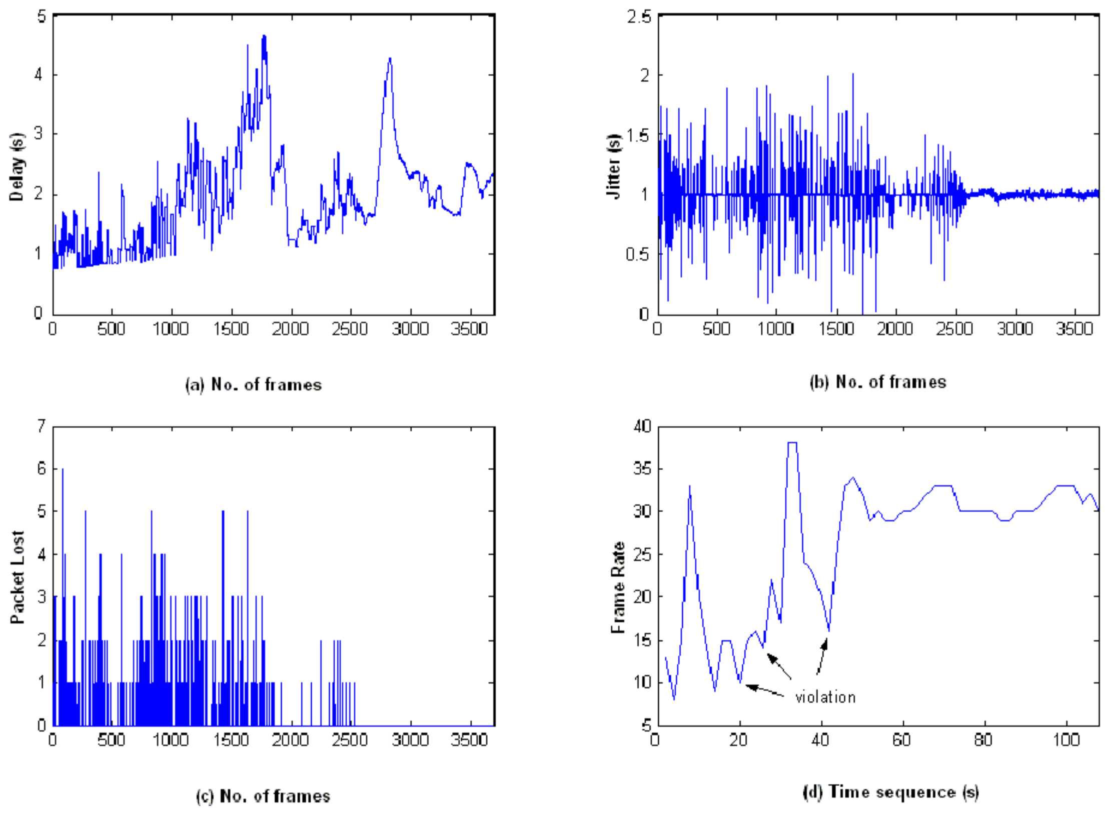


Figure 7-8: Observation of end-to-end QoS w/ and w/o network congestion

After that, we simultaneously launch the CPU utility program and the traffic generator and view their combined impacts on video streaming. Through analyzing end-to-end QoS statistics, we have found that the newly observed QoS pattern is different from those in the previous individual tests. Hence we enlist it as a new violation type which marks the co-existence of the above two violations. This observation coincides with our previous argument that different QoS violations will present different fingerprints in terms of end-to-end QoS exhibition. Hence a synthesized violation will show distinct fingerprint that differ from any individual one.

Thirdly, we have tested our scheme on an open campus network. We deliver audio and video flows in both wired-line and wireless environments. The QoS violation observation in the wired-line environment is similar to that in the testbed environment. Figure 7-9 shows the behavior of video streaming in a 54Mpbs wireless LAN environment. The stream sender and receiver are separated by 4 hops (routers) as we have identified using Tracert command[9] of the Windows XP operating system. We conduct the experiment in late night so that QoS violations due to excessive user competition can be minimized. Hence the packet loss and performance degradation as shown in Figure 7-9 is primary due to wireless link error. It is interesting to see that jitter happens at the transmission of the 400th video frames (Figure 7-9 (b)) and meanwhile packet loss is observed (Figure 7-9 (c)). However, the end-to-end streaming is not seriously affected and no violation is observed at the application level (Figure 7-9 (d)) at the same time. This is possibly because the network jitter is well smoothed at the streaming receiver by techniques such as media buffering. This observation coincides with our previous statement that a drop of value in single low level parameter (i.e., packet jitter) does not necessarily indicate an application QoS

[9] Tracert is a Windows based command-line tool that can be used to trace the path that an Internet Protocol (IP) packet takes to its destination from a source.

violation since such a variation may be compensated by other parts of the end-to-end system.

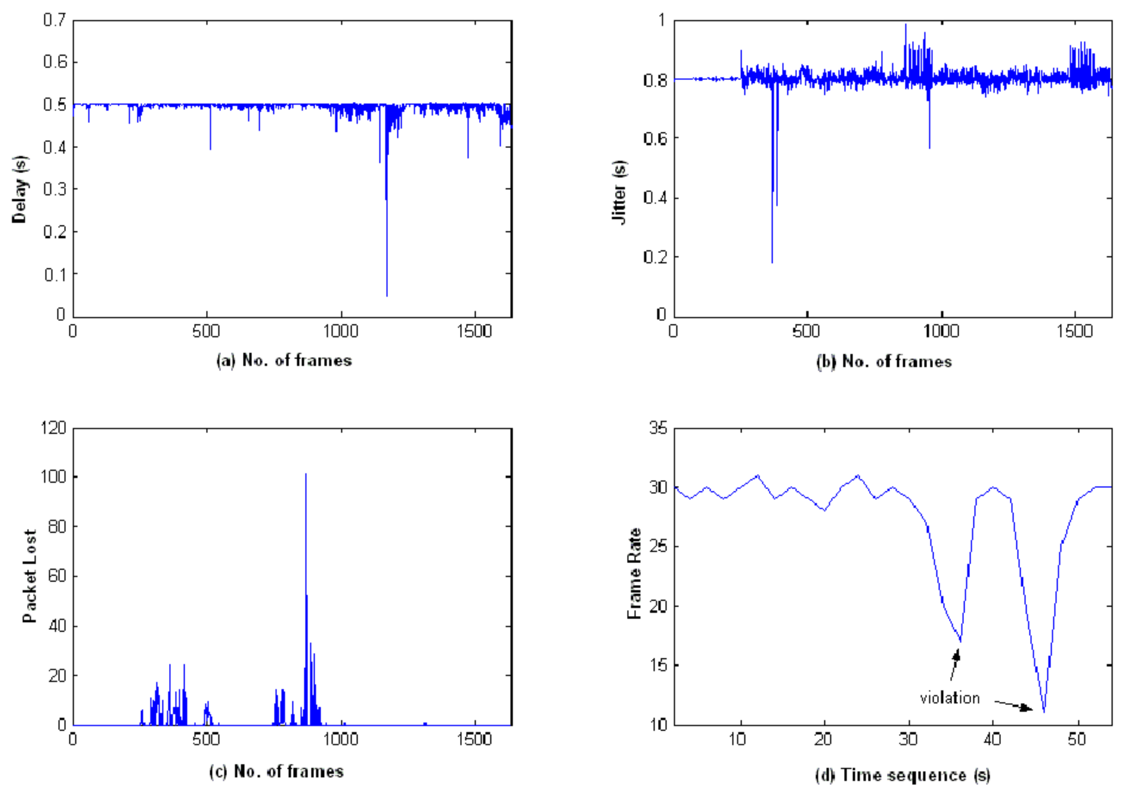


Figure 7-9: Observation of end-to-end QoS variation in wireless communication

Lastly, we have repeated the same CPU violation and network congestion experiments on the PlanetLab platform. We send audio/video flows to PlanetLab nodes located at different areas of the world (Table 7-3) and observe their performance in the presence of QoS violations. The purpose of this series of tests is to assess the correctness of our end-to-end QoS violation diagnosis scheme in a real Internet environment that has complex network dynamics. We similarly collect QoS violation data in these experiments and combine it with that gathered in testbed and wired campus network tests for the subsequent data analysis under the category of wired-line data.

| Media source | Destination PlanetLab nodes | Hops | TTL (ms) |
|---|---|---|---|
| zhoulifeng.ddns.comp.nus.edu.sg | planetlab1.singaren.net.sg | 15 | 250 |
| vivijane.ddns.comp.nus.edu.sg | planetlab1.iii.u-tokyo.ac.jp | 22 | 410 |
| zhoulifeng.ddns.comp.nus.edu.sg | planetlab1.xeno.cl.cam.ac.uk | 22 | 262 |

Table 7-3: QoS violation test with PlanetLab nodes (source from NUS)

### 7.3.2 Data Analysis

We have engaged several neural network algorithms to analyze the end-to-end QoS violation data. These algorithms, according to their training speed and classification speed, can be summarized into online and offline categories as has been discussed in Section 5.4. We will first show that online algorithms can achieve real-time processing capability with high classification accuracy that is comparable to offline algorithms. They are hence more suitable for runtime QoS violation analysis. Next, we will show that the orthonormal algorithm we have proposed and validated in Appendix A presents even better performance compared with other online algorithms.

| Name | No. of Observations | | Attributes | Classes |
|---|---|---|---|---|
| | Training data | Testing data | | |
| Wired-line | 4500 | 3863 | 4 | 4 |
| Wireless | 3000 | 2217 | 4 | 5 |

Table 7-4: Specification of QoS violation Datasets: the wired-line category contains data obtained from testbed, campus network and PlanetLab platform

As explained in Section 5.3, we use both flow descriptors and application QoS parameters for QoS violation classification. We focus on two typical network scenarios: wired-line and wireless environments. For experiments in the wired-line environments (testbed, wired campus network and PlanetLab experiments), we have altogether collected and combined 8363 groups of data, among which 4500 groups are randomly selected as training data. The rest are used as testing data. In classification of QoS violations, flow statistics such as packet delay, jitter, packet loss and frame rate are fed in as the input vector $x$; the output $t$ pinpoints one of the four categories current streaming is undertaking: normal transmission, a sender side CPU violation, a receiver side CPU violation and a congestion violation. Data collected in the wireless environment is used in the performance evaluation of our orthonormal algorithm as will be discussed later. Table 7-4 gives the characteristics of QoS violation datasets. All the input data is normalized into the range [-1, 1]. The activation function we have selected is a sigmoidal function $g(x) = 1/(1+e^{-x})$ for additive neurons. The input weights $a_i$ and hidden biases $b_i$ are randomly chosen from the range [-1, 1]. For SLFN with RBF activation function $\phi(x) = \exp(-\gamma\|x-\mu\|^2)$, the centers $\mu_i$ are randomly chosen from the range [-1, 1] whereas the impact factor $\gamma$ is chosen from the range (0, 0.5).

| **Algorithms** | **Training Accuracy** | | **Training Time (seconds)** | **No. of Neurons** |
|---|---|---|---|---|
| | **Mean** | **Dev** | | |
| ***Offline*** | | | | |
| LMBP | 98.33% | 0.0087 | 423.5 | 30 |
| RANEKF | 92.43% | N.A. | 1.0517e+005 | 495 |
| MRAN | 95.61% | 0.0114 | 5115.5 | 168.4 |
| ***Online*** | | | | |
| Stochastic BP | 92.57% | 0.0249 | 3.8543 | 30 |
| ELM (sigmoid) | 94.26% | 0.0063 | 0.1127 | 30 |
| ELM (RBF) | 94.28% | 0.0053 | 0.1398 | 30 |

Table 7-5: Classification accuracy of QoS violations in different algorithms

Since the offline algorithm RANEKF takes a quite long training time, we only run it once. The results of other algorithms are computed based on 30 trials. Both RANEKF and MRAN automatically determine the number of hidden neurons. For other algorithms in which neuron numbers need to be specified, we have gradually increased the number of neurons from 5 to 50 with the increscent pace of 5 in each round and measure corresponding classification accuracy. For example, the classification accuracy and training time comparison between LMBP and ELM in each tuning round is illustrated in Figure 7-10 and Figure 7-11. Figure 7-10 shows that higher classification accuracy can be achieved with the growth of neuron numbers. However, the required training time also increases as more neurons are employed, which can be seen from Figure 7-11. To balance between accuracy and overhead, we have selected 30 neurons in our experiments for QoS violation classification. Table 7-5 shows the mean training time and accuracy results under such a setting. It can be seen that all these online and offline classification algorithms can achieve high classification accuracy while their training times deviate significantly: online algorithms exhibit much faster training speed compared with offline algorithms. Both Stochastic BP and ELM can finish training of 4000 groups of data within a few seconds and achieve 92% and 94% classification accuracy respectively. Hence these algorithms are suitable for real-time training and diagnosis of QoS violations for their high classification accuracy and short response time. Finally, the training and testing data are randomly generated for each round of data analysis. As a result, every trial may lead to different classification results for the same dataset. The corresponding deviations reflect the stability of testing accuracy in different configurations of training parameters. From Table 7-5 we can find that the deviations of the training classification accuracy in all algorithms are very small which further validates the effectiveness of our neural network approach in QoS violation classification.

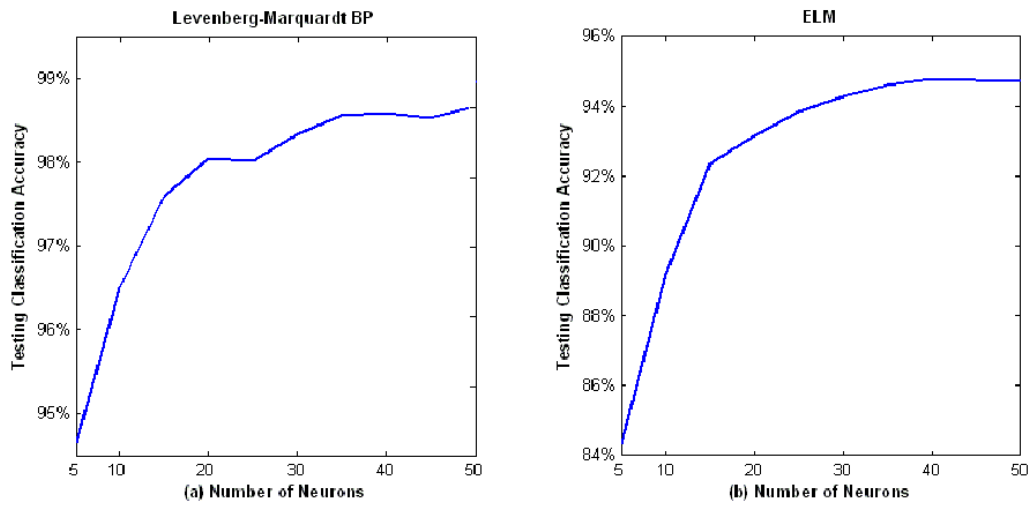


Figure 7-10: Testing classification accuracy comparison between LMBP and ELM

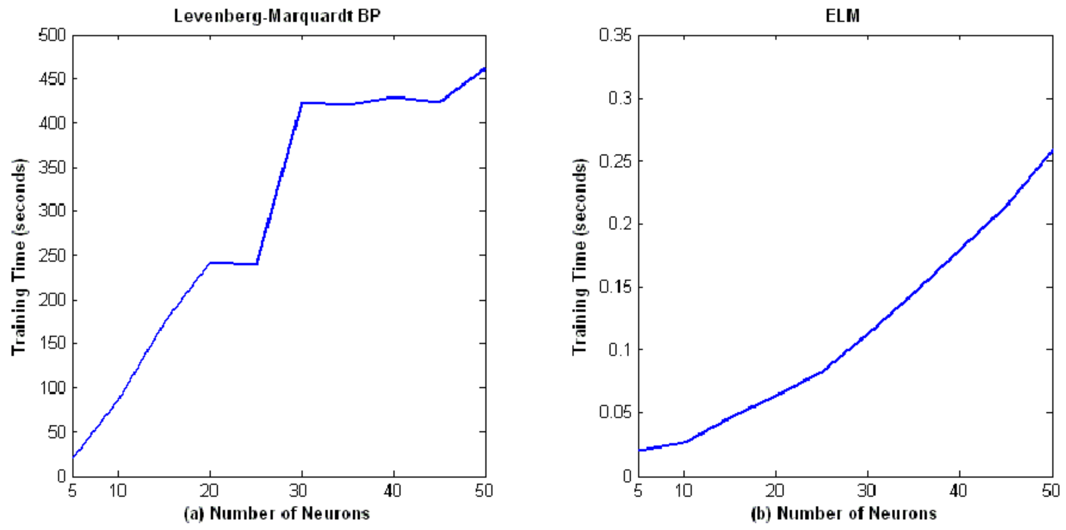


Figure 7-11: Training time comparison between LMBP and ELM

Next, we take a closer look at the performance of our orthonormal algorithm (Appendix A) for QoS violation analysis. For comparison purpose, we use the same set of data for training and testing as in the evaluation of aforementioned algorithms. In addition, we collected 5217 groups of data from the experiment performed within the wireless environment, among which 3000 groups are randomly selected as training data. The rest are again, used as testing data. For wireless QoS test, the structure of the input vector $x$ is the same as that used in the wired-line experiment; the output $t$ refers current streaming to one of the five possibilities: normal transmission, a (sender or receiver) CPU violation, a wireless link error case or a congestion violation. The activation function selected in our orthonormal algorithm is the same sigmoidal function $g(x) = 1/(1+e^{-x})$ for additive neurons. The input weights $a_i$ and hidden biases $b_i$ are randomly chosen from the range [-1, 1]. Other neural network configurations are identical to the aforementioned parameter settings.

| **Name** | **Training Accuracy** | | **Testing Accuracy** | | **Time (seconds)** | **No. of Neurons** |
|---|---|---|---|---|---|---|
| | **Mean** | **Dev** | **Mean** | **Dev** | | |
| Wired-line | 96.37% | 0.0032 | 95.75% | 0.0027 | 0.2737 | 30 |
| Wireless | 87.28% | 0.0041 | 86.60% | 0.0142 | 0.1596 | 30 |

Table 7-6: Classification accuracy for QoS violations in our orthonormal algorithm

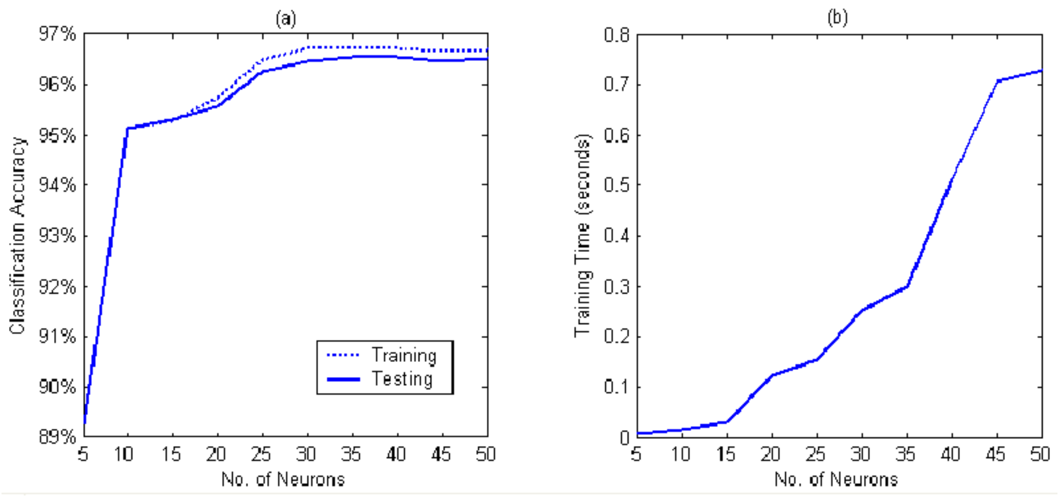


Figure 7-12: Performance of the proposed orthonormal algorithm in QoS violation classification: (a) training and testing accuracy curves, (b) training time curve

The experiment results are similarly computed based on 50 trials. We have gradually increased the number of neurons from 5 to 50 with the increscent pace of 5 in each round and measure corresponding classification accuracy and training time. The data analysis results are shown in Figure 7-12. Figure 7-12(a) shows that higher classification accuracy can be achieved with the growth of neuron numbers. However, Figure 7-12(b) illustrates that the required training time also increases as more neurons are employed. This observation is similar to the corresponding characteristics of other neural network algorithms such as LMBP and ELM. To balance between accuracy and overhead, we also selected 30 neurons in our experiments for QoS violation classification.

Table 7-6 shows the mean training and testing classification accuracy results of our orthonormal algorithm under the setting of 30th neurons. It can be seen that our classification algorithm can achieve high classification accuracy in both wired-line and wireless environments. For QoS violation experiments in wired-line environments, we have a good chance (95.75%) to correctly assert a violation after training. The accuracy of wireless testing is comparatively lower due to other (uncontrollable) impacting factors that blur the margin between, for example, wireless channel error and traffic congestion (e.g., the possible coexistences of two violations). However, such 86.60% accuracy still greatly outperforms the traditional rule-based method as will be explained later. We can also see from the Table 7-6 that the deviations of the training and testing classification accuracy are very small. Finally, it takes less than 0.3 second for our orthonormal algorithm to train around 4500/3000 sets of 4-dimension input data, which is competent for real-time processing. In brief, our orthonormal algorithm can achieve slightly better performance compared with other online neural network algorithms in runtime QoS violation analysis in terms of classification accuracy and training/testing speed.

Through feature reduction analysis, we have found that packet delay and jitter play an important role in identifying a network congestion violation while packet jitter and receiving frame rate is effective in identifying a CPU violation. There is no single QoS parameter which can effectively distinguish all violation cases. To show the advantage of our orthonormal approach over a traditional rule-based method, we have also developed a set of rules to detect QoS violations. Each rule examines either packet delay or packet loss rate to identify a QoS violation and trigger an adaptation action. The violation threshold for each discrimination parameter is defined as its mean value in a violation period (calculated from collected data in previous experiments). We repeat the same experiments using such a rule-based method and find that an overall detection accuracy of only 73% can be achieved. The remaining 27% cases are either (1) no alarm reported in case of violations, or (2) mis-alarm of QoS violations (e.g., a CPU violation in a sender host to be asserted as network congestion). All the above facts show that our orthonormal classification approach presents better results in diagnosis of QoS violations than a traditional rule-based method.

## 7.4 END-TO-END QOS MANAGEMENT

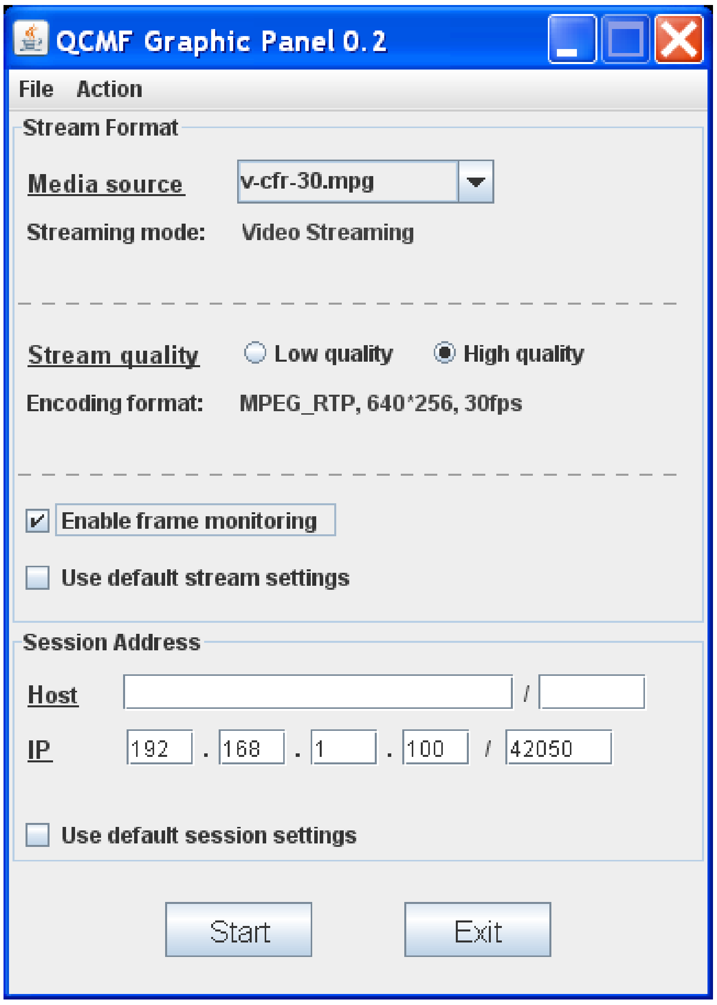
### 7.4.1 QCMF Management Procedures

```
App. Name := Video_transmitter
App. Type : = video_streaming
Original Video Codec : = MPEG
Supported Video Frame Rate : = 20/25/30 fps
Audio channels := mono
Video Delivery Quality := All quality
Text Delivery Quality := reliable/Nil
Supported platform := Windows/Unix
```

Figure 7-13: An example QList for video streaming

The QoS management in QCMF is tested through audio/video transmission between two end-hosts. The testbed configurations have been explained in Section 7.1. At build-time, a QList which records QoS characteristics (Figure 7-13) of the A/V streaming application is read in by QCMF. Such a high level specification from an application designer is enriched by the QoS domain specification held by QCMF and is subsequently translated into AQoSpec, which in turn is mapped into middleware and network settings. Here middleware configuration mainly refers to the composition of protocol stacks and network configuration refers to communication parameters settings such as socket addresses. We use a Linux router of two traffic queues (i.e., FIFO queue and RED queue) as a generic model to emulate a QoS-enabled subnet of two service classes. Before streaming, QoS negotiation is carried out between the media sender and receiver to reach agreement of end-hosts' protocol stack settings and communication parameter configurations (e.g., socket address, synchronization timeout value).

The GUI interface for user control at the sender side is illustrated in Figure 7-14 where an end-user can manually configure streaming settings such as the media file to stream, streaming quality, monitoring facilities and session addresses. Alternatively, an end-user can choose to use default setting so that all relevant communication settings will be configured automatically.



Stream Format

Media source: v-cfr-30.mpg

Streaming mode: Video Streaming

Stream quality: ○ Low quality [● High quality]

Encoding format: MPEG_RTP, 640*256, 30fps

☑ Enable frame monitoring

☐ Use default stream settings

Session Address

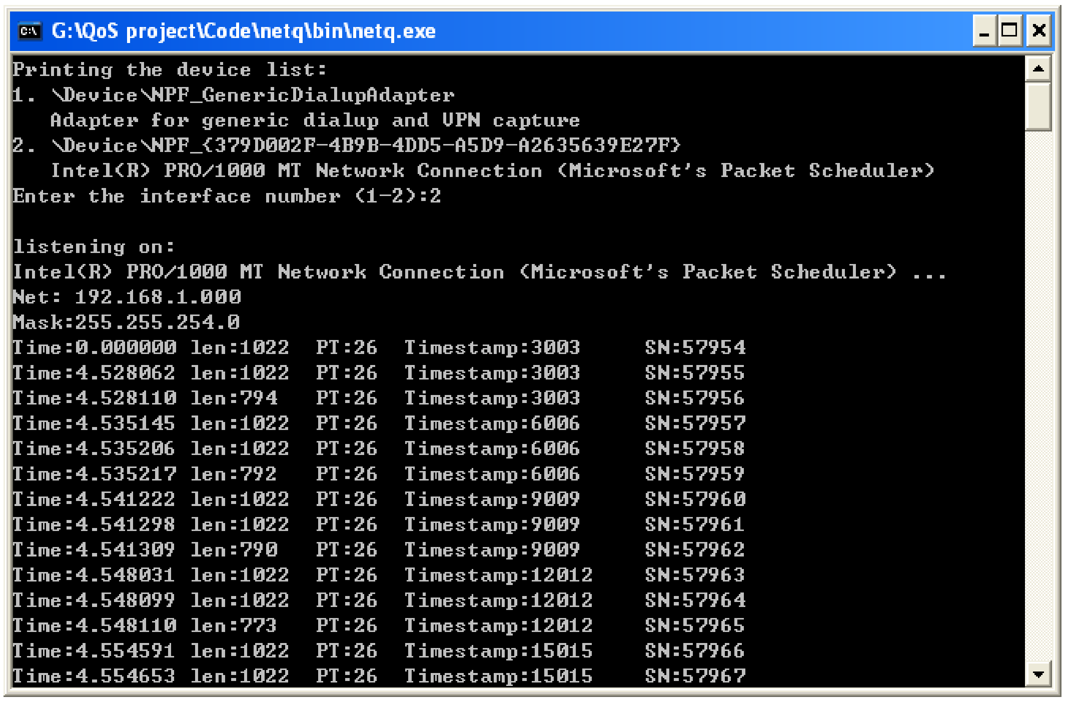
Host: /

IP: 192 . 168 . 1 . 100 / 42050

☐ Use default session settings

Figure 7-14: Graphic User Interface (GUI) for streaming

During streaming, QCMF continuously detect QoS violations at the media receiver. As explained, this is achieved through monitoring and comparing flow data at the NIC of the receiver (Figure 7-15). Information to be collected includes receiving time of a RTP packet, RTP payload type, RTP timestamp and sequence number. The receiving time of each RTP packet is extracted to calculate flow level traffic statistics such as packet delay, jitter and throughput. RTP payload type indicates the format of the multimedia content (e.g., 26 represents JPEG) which has been standardized in [59]. The RTP timestamp and sequence number are, as explained in Section 5.3.3, used to restore the sequence of RTP packets in assembling application frames.



```
Printing the device list:
1. \Device\NPF_GenericDialupAdapter
   Adapter for generic dialup and VPN capture
2. \Device\NPF_{379D002F-4B9B-4DD5-A5D9-A2635639E27F}
   Intel(R) PRO/1000 MT Network Connection (Microsoft's Packet Scheduler)
Enter the interface number (1-2):2

listening on:
Intel(R) PRO/1000 MT Network Connection (Microsoft's Packet Scheduler) ...
Net: 192.168.1.000
Mask:255.255.254.0
Time:0.000000 len:1022  PT:26  Timestamp:3003     SN:57954
Time:4.528062 len:1022  PT:26  Timestamp:3003     SN:57955
Time:4.528110 len:794   PT:26  Timestamp:3003     SN:57956
Time:4.535145 len:1022  PT:26  Timestamp:6006     SN:57957
Time:4.535206 len:1022  PT:26  Timestamp:6006     SN:57958
Time:4.535217 len:792   PT:26  Timestamp:6006     SN:57959
Time:4.541222 len:1022  PT:26  Timestamp:9009     SN:57960
Time:4.541298 len:1022  PT:26  Timestamp:9009     SN:57961
Time:4.541309 len:790   PT:26  Timestamp:9009     SN:57962
Time:4.548031 len:1022  PT:26  Timestamp:12012    SN:57963
Time:4.548099 len:1022  PT:26  Timestamp:12012    SN:57964
Time:4.548110 len:773   PT:26  Timestamp:12012    SN:57965
Time:4.554591 len:1022  PT:26  Timestamp:15015    SN:57966
Time:4.554653 len:1022  PT:26  Timestamp:15015    SN:57967
```

Figure 7-15: Netq program for data packet capturing at the media receiver

A QoS violation in QCMF is handled by end-to-end evaluation and coordination as described in Chapter 6. An adaptation solution will be selected by the flow receiver in view of policies and end-to-end QoS status. Such an adaptation could be a service upgrade in the network or an end-host solution that tries to maintain the transmission quality. Some of the end-host adaptation strategies that are defined in QCMF for video streaming are illustrated in Table 7-7, each of which is activated when its end-to-end conditions is met (e.g., packet loss rate of the application or CPU availability of a host machine). Adaptation actions to be taken include changing communication components (e.g., replace UDP with TCP protocol in the protocol stacks), modifying configuration parameters (e.g., increase receiving buffer size), or even reduce application QoS (e.g., resize picture dimension).

| Problems | Observed End-to-end QoS States | Actions |
|---|---|---|
| Frame rate too low | Many lost packets and low frame delay | Selective retransmission (e.g., UDP -> TCP) |
| | Many lost packets and high frame delay | Forward error correction & smaller picture |
| End-to-end delay too high | Low network bandwidth availability and low CPU load | Higher compression ratio |
| | Low network bandwidth availability and high CPU load | Smaller picture size |

Table 7-7: User-defined adaptation policies for video streaming

### 7.4.2 QCMF Management Performance

Subsequently, we have designed experiments to measure the latency of QCMF in different phases of QoS operations, ranging from build-time negotiation to runtime QoS adaptation. Time taken for various QoS management functions is shown in Table 7-8. As can be found out, QCMF exhibits a one time initiation delay around 156 ms at session initiation stage, which is mainly used to setup the ontology model for QMan middleware, read QoS requirements from applications (i.e., an I/O operation), and configure end-to-end QoS settings through signaling. The overhead for knowledge preparation, as explained, is incurred by Jena to read in and analyze the middleware ontology knowledge base which contains ontology definitions about protocols (250 ms) and to perform knowledge reasoning (1313 ms). Start-up training of QoS violation data absorbs another 252 milliseconds. After that, a QoS violation can be identified within 10 milliseconds at runtime. The control channel between sender and receiver is established using Java RMI registry technology which is a server-side name service that allows RMI clients to get a reference to the server object. In QCMF, this RMI registry is used for the SyncMaster (at the receiver side) to announce its presence and for the SyncSlave (at the media sender) to retrieve and register with it. The creation of these two components at end-hosts takes 453 milliseconds. The signaling among them, however, takes only 6 ms since the propagation delay in Ethernet is really short.

| Operations | | Time (ms) |
|---|---|---|
| Instantiation of QCMF | | 156 |
| Creation of synchronization objects | | 453 |
| Knowledge Preparation (total) | | 1563 |
| | *Creation of reasoning model* | 1313 |
| | *Compilation of application QoS ontology* | 250 |
| Start-up QoS Violation Data Training | | 252 |
| Control channel signaling | | 6 |
| Protocol Stack Composition (total) | | 1446 |
| | *Discovery of protocol components* | 93 |
| | *Form of candidate protocol stacks* | 3 |
| | *Check stack validation* | 31 |
| | *Creation of functional protocol stack* | 103 |
| QoS Adaptation through Protocol Stack Re-composition (total) | | 948 |
| | *Violation Analysis* | 10 |
| | *End-to-end evaluation* | 272 |
| | *Protocol stack re-negotiation* | 11 |
| | *Validate new stack composition* | 29 |
| | *AC stack adaptation* | 112 |

Table 7-8: Time taken in end-to-end QoS management

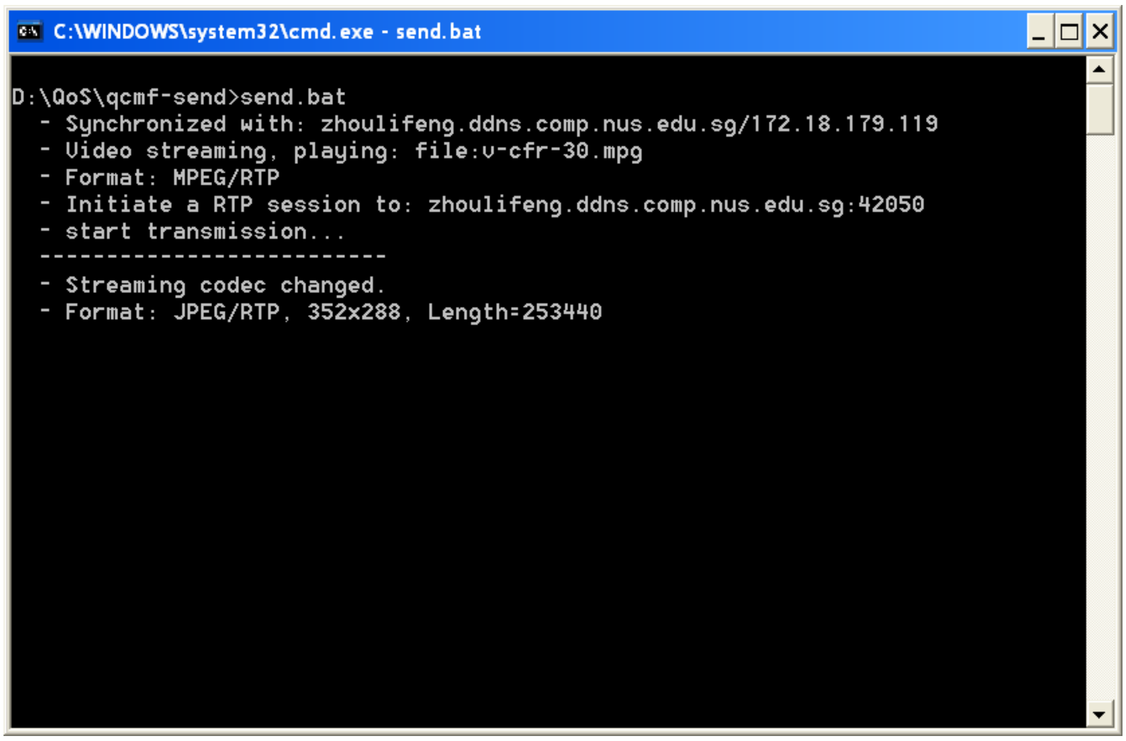
In QCMF, the semantic discovery of appropriate communication protocols is performed by issuing an SPARQL query on protocol knowledge base, an example of which is shown in Figure 7-16. Such a SPARQL language [150] is also used to check the consistency and dependence among protocol entities. After the knowledge preparation stage just described, we have found that each query can be answered at an average rate of a few tens of milliseconds.

```
ns = "<http://www.comp.nus.edu.sg/qcmf/qNegotiation#>";
rdfs = "<http://www.w3.org/2000/01/rdf-schema#>";
prefix = "PREFIX rdfs: " + rdfs + " PREFIX ns: " + ns + " ";

String qservice = prefix
    + "SELECT ?service"
    + "WHERE {?service ns:serviceinstance \"" + ns:os\""."
    + "ns:os + \"ns:type + ns:UNIX.}"

Query query = QueryFactory.create(qservice);
```

Figure 7-16: Sample SPARQL query for ontology integrity check between two instance classes

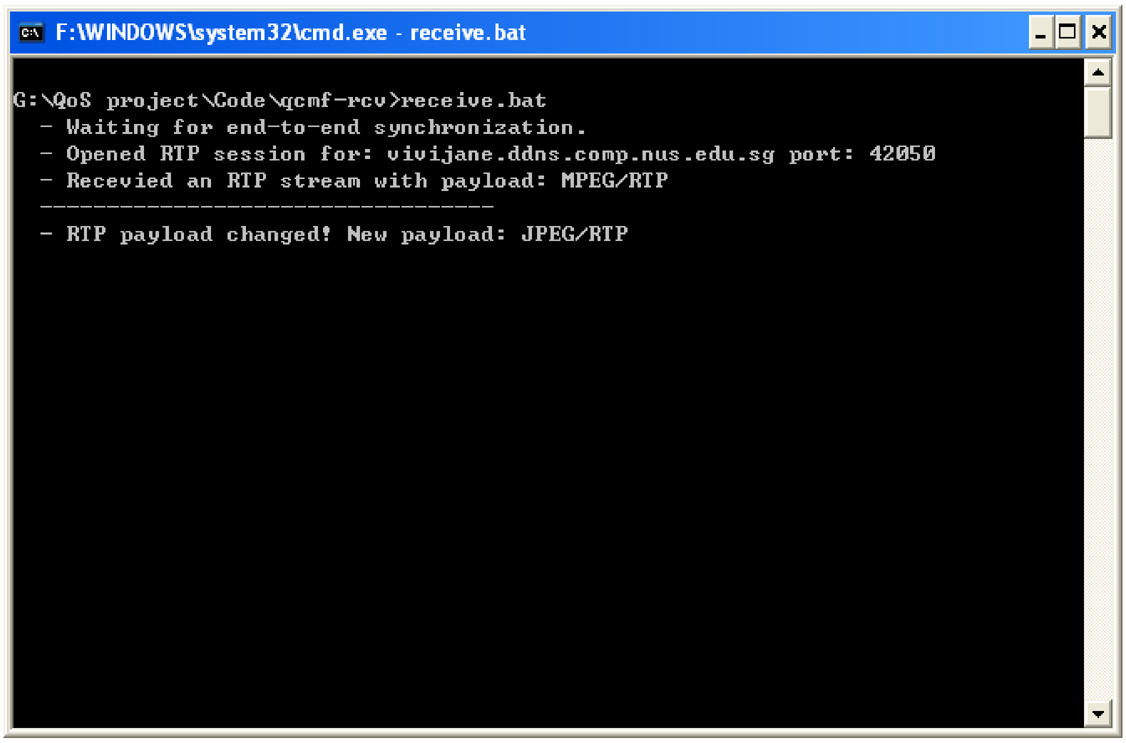
Runtime QoS adaptation in QCMF may take place at either application level, middleware level or network level, depending on system policies (refer back to Section 6.4.2 for details) and end-to-end evaluation results. A middleware level adaptation takes nearly 1 second, which includes the time to determine a violation (20 ms), evaluate a solution (272 ms) and perform a middleware adaptation (312 ms). Alternatively, if a network QoS upgrade solution is been selected, the adaptation will also take around 1 second, in which 7 milliseconds is used for control signaling and the rest is for the adaptation action performed in the Linux router. The time taken for application level adaptation largely depends on the business logic of multimedia applications. In the case of our video streaming application, the action to reduce video resolution and restart transmission takes around 2 seconds.

The effect of middleware adaptation (i.e., protocol stack swapping) in case of network congestion is illustrated in Figure 7-19. A video stream is delivered from the streaming sender (Figure 7-17) to the receiver (Figure 7-18). Upon detection of the congestion violation, a protocol stack swapping adaptation is enforced which switches the video stream from original MPEG format to JPEG format. Figure 7-19 shows the variation of video frame rate before and after the violation.



```
D:\QoS\qcmf-send>send.bat
  - Synchronized with: zhoulifeng.ddns.comp.nus.edu.sg/172.18.179.119
  - Video streaming, playing: file:v-cfr-30.mpg
  - Format: MPEG/RTP
  - Initiate a RTP session to: zhoulifeng.ddns.comp.nus.edu.sg:42050
  - start transmission...
  --------------------------
  - Streaming codec changed.
  - Format: JPEG/RTP, 352x288, Length=253440
```

Figure 7-17: Stream delivery and adaptation at the media sender



```
G:\QoS project\Code\qcmf-rcv>receive.bat
  - Waiting for end-to-end synchronization.
  - Opened RTP session for: vivijane.ddns.comp.nus.edu.sg port: 42050
  - Recevied an RTP stream with payload: MPEG/RTP
  ----------------------------------
  - RTP payload changed! New payload: JPEG/RTP
```

Figure 7-18: Stream receipt and adaptation at the media receiver

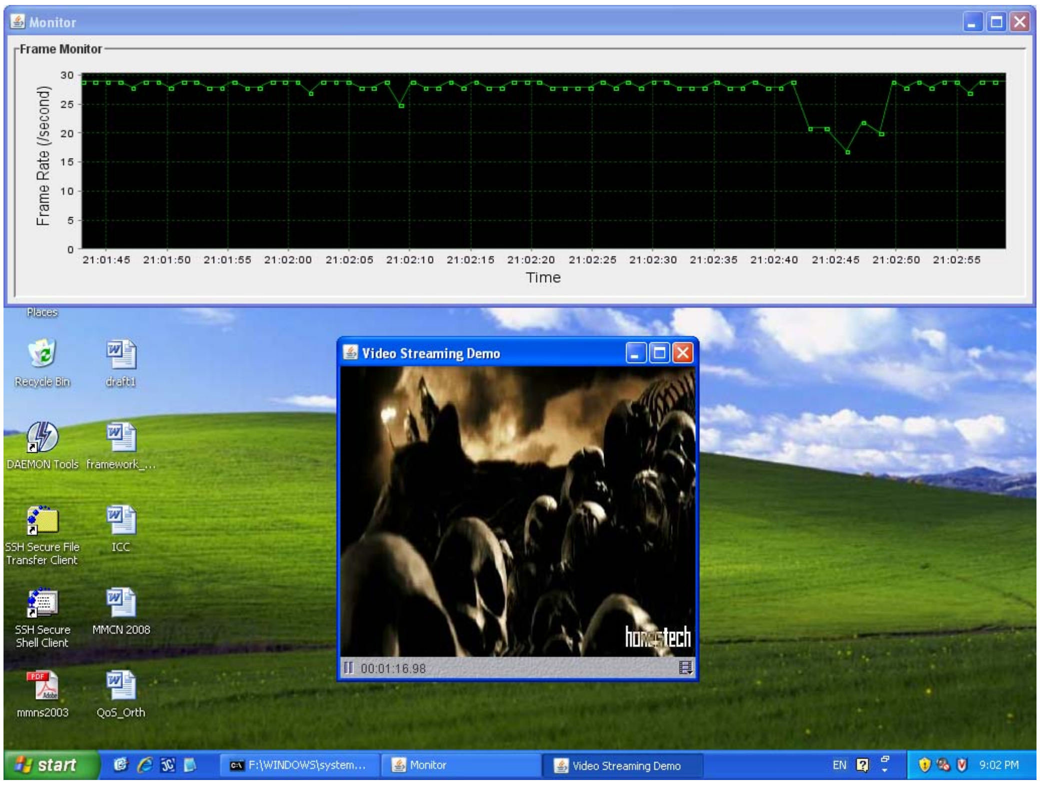


Figure 7-19: Quality fluctuation of the receiving frame rate before, during and after congestion violation

We have also conducted experiments to observe QoS violations in audio streaming. Audio streaming typically requires much less resources (CPU time slice and network bandwidth) compared with video streaming. Unlike video streaming whose throughput may vary significantly from time to time (i.e., variable bit rate), audio streaming often produce traffic of relatively constant rate. Hence the monitoring and analysis of audio streaming in QCMF is much simpler than that in video streaming since the fingerprint of a violation can be easily distinguished from that of a normal transmission (even by eyes). An observation of network congestion in MPA_RTP audio streaming is recorded in Figure 7-20. The throughput of the audio stream is around 120Kpbs. We have limited the link bandwidth to 200 Kbps and launched background traffic of constant rate 100 Kbps. As a result, network congestion takes place at the Linux router and packet loss is observed in audio streaming. Such a congestion violation can be easily detected (since the violation traffic pattern is consistent) and solved by QCMF through either (1) swapping to a less bandwidth demanding audio codec such as G723. or (2) shifting to another network service class (router queue). The last picture of Figure 7-20 compares the audio throughput at the flow sender and receiver to show the effect of network congestion on end-to-end audio flow throughput. On the other hand, we find that CPU violation has bare effect on audio streaming quality. This is probably because audio streaming consumes little CPU time slice and hence is not quite CPU-intensive.

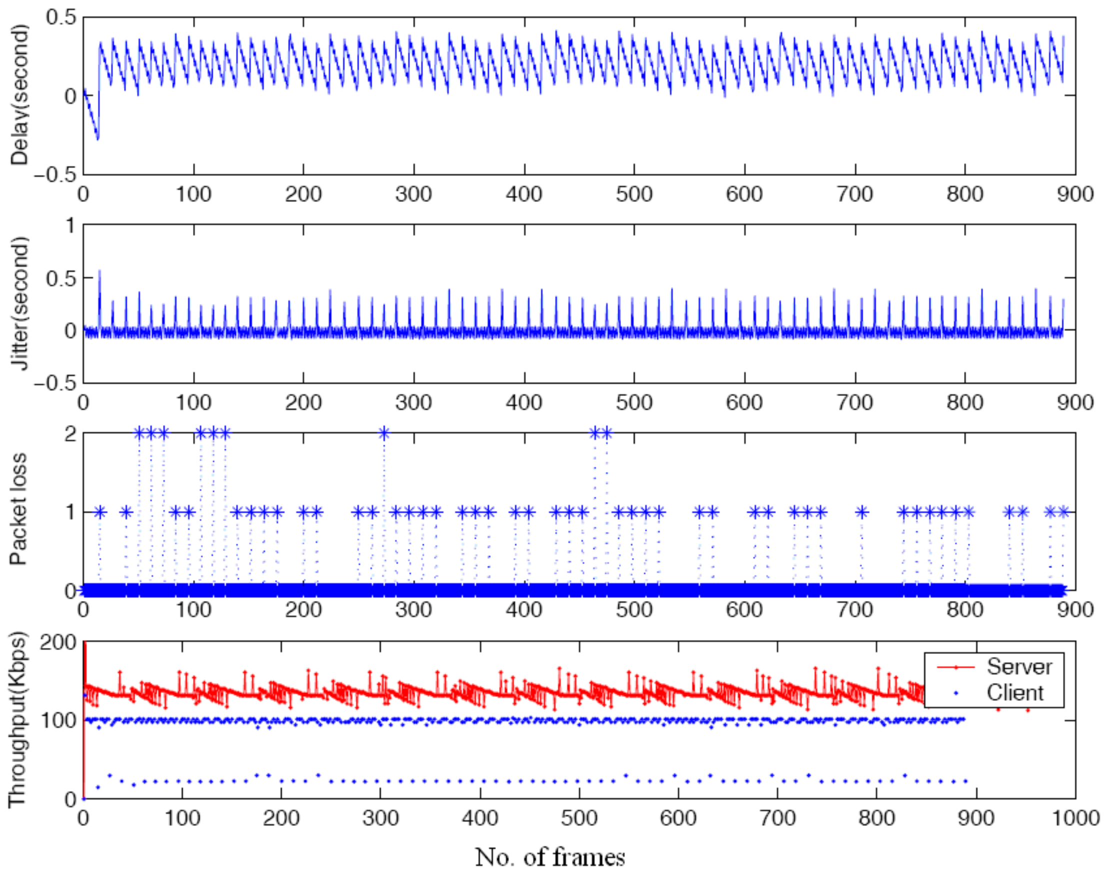


Figure 7-20: End-to-end flow statistics of an audio streaming under violation

### 7.4.3 Control Channel Overhead

As mentioned in Section 4.5.1, the control channel among end-to-end QoS components is largely established using Java RMI technology. To illustrate the overhead of RMI signaling in a large scale network, a series of measurements have been done whereby a client invokes a remote method and passes difference sizes (16, 32, 64 … 32768bytes) of control message to the sender. As is shown in Figure 7-21, the first three series (lower part) were performed in a normal end-to-end transmission (without QoS violations) under 10Mbps, 100Kbps and 50Kbps environments respectively. The connection bandwidth other than 10Mbps is emulated by running background traffic that occupies certain amount of link bandwidth. Another four series of experiments (upper part) were performed with the presence of end-to-end QoS violation. For instance, Violated1 was recorded in a 100Kbps environment where the end-to-end media flow contends with another network flow (produced by the traffic generator program) of 129Kbps. According to the experiment results, RMI does not significantly contribute additional delay to QoS functions at binding-time. As the largest control message is set to 4Kbytes, RMI produces delay that is less than 10ms (under a 10Mbps network environment and without QoS violation). In the presence of QoS violations, RMI incurs slightly higher delay because the control message shares the same link with the data plane (i.e., FIFO queue at the router). However, the size of the control message in a practice is not likely to exceed 1Kbytes, hence the overall delay of RMI in both violation and non-violation environments are still acceptable for real-time QoS processing.

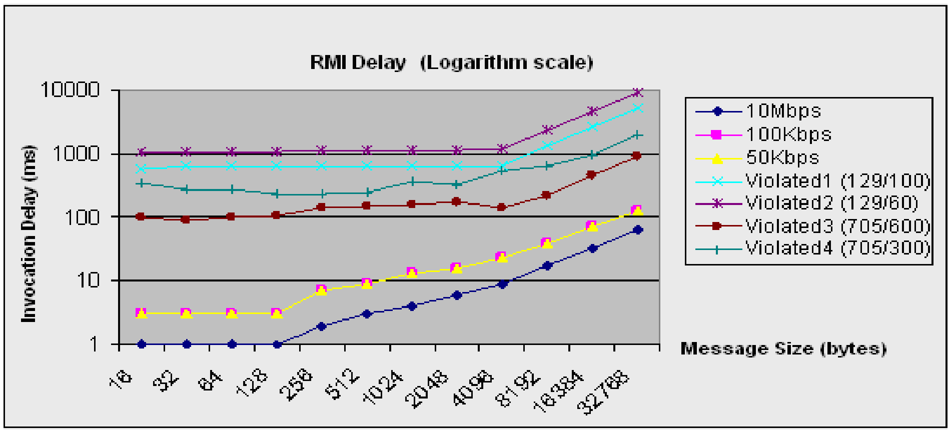


Figure 7-21: RMI invocation delay for the control plane (Logarithm scale)

## 7.5 SUMMARY

A QCMF prototype has been implemented on J2SE platform. Experiments have been done to evaluate respective functions of QCMF and measure their performance. These include the knowledge processing overhead, QoS violation detection efficiency (accuracy and latency) and overall end-to-end QoS management performance. Experiment results show that QCMF does not introduce unacceptable overhead while it offers flexibility to manage QoS in an end-to-end manner. On the other hand, we observe that ontology processing a relative time-consuming process, which may not be appropriate for resource-constrained devices such as PDA or handset. Hence, further research could be conducted in the future to introduce a set of proxies to take over the ontology processing task in the case of mobile multimedia streaming.

# CHAPTER 8
# CONCLUSIONS AND FUTURE WORK

This chapter concludes the thesis with a summary of our research contributions and discussion of topics for future work.

## 8.1 THESIS SUMMARY

Today more and more multimedia applications are becoming QoS-aware, thus presenting predictable, configurable and adaptable behaviors. For over a decade the Internet engineering and research community has debated, designed, and ignored IP QoS tools and techniques. Some of these techniques, such as DiffServ and RSVP have shown to be very valuable to QoS-sensitive applications (such as audio and video transmission). However, till today these solutions are yet to be widely endorsed by network service providers or the users. There is an urgent sense that something might be needed, despite the fact that some of the impeding forces working against a wider deployment of these technologies are non-technical, but has become very important consideration since Internet become a commercial commodity.

In this thesis, we have analyzed and found several shortcomings in existing QoS research which may delay the prevailing deployment of QoS as a utility service. Firstly, layered QoS provision is not a sustainable solution to end-to-end QoS, as has been shown in various research prototypes. Although each of those projects is keen in its respective area, end-to-end QoS can hardly be achieved. This is simply because the performance of a multimedia application relies on both end-host and network resources, which suggests that QoS management in all these areas is essential. Secondly, most end-to-end schemes emphasize on the design of end-to-end QoS-flow and service composition. They have not considered the distributed and collaborative nature among end-to-end QoS sub-systems. On the other hand, a few end-to-end projects have designed a chain of new QoS services in each area of end-to-end QoS provision. Although such a solution may achieve better efficiency due to its customized and seamless design from network layer to application layer, it is not an attractive proposition as the cost to replace existing Internet architecture with a brand new set of technologies.

In view of the above limitations, we propose a semantic-based QoS coordination and management framework (QCMF) for a cooperative end-to-end QoS delivery. QCMF aims to accommodate QoS services at different system layers and manage their interactions for the benefit of end-to-end QoS delivery. Our solution embraces existing and new QoS mechanisms at three entity levels: the network level, the middleware level and the application level. We treat each of these QoS facilities as an end-to-end QoS component and model its capabilities and behavior with respect to the requirements of end-to-end QoS coordination. A knowledge plane is thus established which records QoS information of each end-to-end component. Encoded in RDF/RDFS, the knowledge plane can grasp the relationships among QoS entities within and among QoS components. Meanwhile, a uniform interface is provided to different users of an end-to-end QoS system (e.g., network service providers and multimedia applications), which facilitates the sharing and machine processing of QoS knowledge. Capitalizing on the knowledge plane, the control plane manages the transmission quality of multimedia applications whereby (1) QoS knowledge is shared either on a regular basis (push mode) or upon request (pull mode), (2) QoS signaling and adaptation is fulfilled in a separate channel so that it will not affect the performance of data transmission, (3) decision-making respect to QoS configurations (e.g., network service level selection, communication parameter determination) and QoS adaptation is done in a pure end-to-end manner so as to avoid the scalability problem in network QoS management. Other outstanding research features of QCMF are listed below.

1. At application level, QCMF allows applications to define QoS requirements either in a precise way or in a “fuzzy” way. In the latter approach, application programmers do not need to fix the name of QoS services to be engaged at design stage. Instead, the necessary features of the desired service are to be described. Semantic mapping will be performed by SQS, an application level service module of QCMF, which translates application requirements into suitable runtime QoS configurations. Such an approach enjoys such an advantage that flexible component-based service composition can be achieved at build-time. In addition, a QoS service that fails to satisfy application requirements can be transparently replaced by another component of similar functionality and QoS performance at runtime.

2. At middleware level, QCMF provides one possible dimension of QoS adaptation within the communication protocol stack. Directed by QMan, a middleware module of QCMF, protocol stack composition and re-composition can be achieved to cope with diverse runtime conditions and resource availability. To detect a QoS violation, QMan monitors the traffic pattern of an end-to-end flow and recognizes a violation once a similar violation pattern repeats. Upon occurrence of a QoS violation, an adaptation choice is made by QMan as the outcome of a coordinated decision-making process

among three dimensions (i.e., applications, middleware and networks) in view of the available end-to-end adaptation choices and administrative policies.

3. At network level, QCMF defines a generic QoS model for QoS service differentiation which can be easily mapped to existing standard network QoS models such as DiffServ which has the same basic QoS disciplines. In this generic model, we define several QoS concepts and principles that are pertinent to the precise description of a network QoS service and the collaboration among end-host QoS and network QoS. These concepts include service class, service degradation path, adaptation cost, adaptation rejection probability and utility factor. A coordinated adaptation decision-making is made by QCMF in a pure end-to-end manner through information sharing and control signaling in light of various end-to-end QoS heuristics and policies.

## 8.2 Future Work

There are a variety of avenues for the future work, including:

*Prediction mechanisms for QoS violations*

A possible extension that helps to minimize the impact of QoS violations on multimedia transmission is to predict the appearance of a QoS violation before it severely harms an end-to-end transmission. At current stage, we validate through experiments that a type of QoS violation will present a consistent fingerprint in terms of the observed application performance and end-to-end traffic pattern. Based on such an observation, we may further hypothesize that similar flow traffic pattern is likely to repeat “on the eve of” the happening of the same violation. By observing such a pattern, prediction of QoS violations may becomes possible which eliminates a QoS violation just at its beginning.

*Supervised learning vs. unsupervised learning*

Our QoS violation analysis scheme is currently bounded by the supervised machine learning method we have chosen. For each kind of QoS violation, a violation class need to be setup beforehand and corresponding training data be collected. Hence we are not able to tell the appearance of a new QoS violation type which has never popped up before. In that case, the neural network learning algorithm will mistakenly classify a new QoS violation phenomenon into one of the known violation types. To solve this problem, we will probably look into more flexible unsupervised learning method such as clustering techniques. Clustering is the process of grouping objects on the basis of perceived similarities. Many clustering models and algorithms allow the number of clusters to vary with problem size and let the user control the degree of similarity between members of the same clusters by means of a user-defined constant [151]. Hence the employment of clustering techniques would probably give more flexibility in our QoS violation classification.

*Refinement of cross-component QoS adaptation scheme*

As described in Chapter 6, we have established models for QoS description of end-hosts and networks and defined primitives to facilitate their interaction. Residing at the end-hosts, an evaluation and adaptation algorithm runtime selects a network QoS component for adaptation based on the estimated utility factor. The preliminary simulation results validate the design of such a collaboration scheme. In the future, we plan to further research into the interaction among various QoS components. For example, we are currently investigating into the scenario that multiple QoS components suffer from QoS violations simultaneously and hence more complex signaling is required to fulfill the coordination. We tend to solve this problem by deriving coordination models that mathematically describes the inter-operations among QoS components, relying possibly on control theory.

*Performance evaluation over the global Internet involving multiple QoS domains*

Our experiments at current stage are mainly conducted on our testbed. Besides, we have tested the performance of our QoS violation detection scheme on the university network (both wired-line and wireless network). Similar experiments have also been conducted on the PlanetLab platform to test the functionalities of our end-to-end management system, as has been explained in Section 7.3.

To further study our prototype system in the Internet environment, we are looking into the possibility to work with large network service providers or router companies to comprehensively test the functions and performance of our QCMF implementation in a large scale network. Ideally, such a testing environment should be a heterogeneous QoS-aware environment where our end-to-end coordination and adaptation scheme could be deployed and its performance be physically measured.

# APPENDIX A
# ORTHONORMAL NETWORK FOR CLASSIFICATION

In this appendix, we will prove that an orthonormal network is universal approximation and hence can be used for classification. We will also extend the scope of hidden neurons from kernel functions to additive functions, the latter of which is, as said, of higher classification accuracy. Our orthonormal algorithm for QoS violation classification will be summarized in Section D based on the knowledge of orthonormal basis and Gram-Schmidt orthonormalization method for construction of it presented in Section B and C respectively.

## A. SINGLE HIDDEN LAYER FEEDFORWARD NETWORK WITH RANDOM HIDDEN NODES

We first introduce some terminologies and background information for later discussion of our main results. The output of a standard single hidden layer feedforward network (SLFN) with $L$ hidden nodes can be represented by $f_L(x)=\sum_{i=1}^{L}\beta_i g(a_i,b_i,x)$, where $a_i$ and $b_i$ are learning parameters of hidden neurons and $\beta_i$ is the weight connecting the $i^{th}$ hidden neuron to the output neurons; $g(a_i,b_i,x)$ is the output of the $i^{th}$ hidden neuron with respect to the input $x$. From the network architecture point of view, two main SLFN network architectures have been investigated, namely additive neurons and kernel neurons. For additive neurons, the activation function $g(x):\Re\rightarrow\Re$ takes the form $g(a_i, b_i, x) = g(a_i \cdot x + b_i)$. For kernel neurons, the activation function $g(x): \Re \to \Re$ takes the form $g(a_i, b_i, x) = g(b_i \|x - a_i\|)$, where $b_i \in \Re^+$ is a positive real value.

For a series of $N$ arbitrary distinct training samples $(x_i, t_i)$, $i = 1, \cdots, N$, $x_i = [x_{i1}, x_{i2}, \cdots, x_{in}]^T \in \Re^n$ is an input vector and $t_i = [t_{i1}, t_{i2}, \cdots, t_{im}]^T \in \Re^m$ is a target vector. A standard SLFN with $L$ hidden neurons and activation function $g(x)$ can be expressed as

$$\sum_{i=1}^{L} \beta_i g(a_i, b_i, x_j) = o_j, j = 1, \cdots, N ,$$

where $o_j$ is the actual output of SLFN.

A standard SLFN with $L$ hidden neurons can learn $N$ arbitrary distinct samples $(x_i, t_i)$, $i = 1, \cdots, N$, with zero error, meaning that there exist parameters $a_i$ and $b_i$, for $i = 1, \cdots, L$, such that

$$\sum_{i=1}^{L} \|o_i - t_i\| = 0 .$$

Therefore our ideal objective is to find proper parameters $a_i$ and $b_i$ such that

$$\sum_{i=1}^{L} \beta_i g(a_i, b_i, x_j) = t_j, j = 1, \cdots, N .$$

The above $N$ equations can be expressed as

$$H\beta = T$$

where $\beta = [\beta_1, \cdots, \beta_L]^T$, $T = [t_1, \cdots, t_N]^T$ and

$$H = \begin{bmatrix} g(a_1,b_1,x_1) & \cdots & g(a_L,b_L,x_1) \\ \vdots & \ddots & \vdots \\ g(a_1,b_1,x_N) & \cdots & g(a_L,b_L,x_N) \end{bmatrix}_{N\times L} .$$

The matrix **H** is called the hidden layer matrix of the SLFN.

Before we present the universal approximation result, we now introduce the following lemma (as has been proved in [137]) for construction of an orthogonal basis with random hidden neurons.

**Lemma 1** *Given a standard SLFN with N hidden neurons and an activation function* $g(x)$ *which is infinitely differentiable in any interval, for N arbitrary distinct samples* $(x_i,t_i)$*, where* $x_i \in \Re^n$ *and* $t_i \in \Re^m$*, for any* $a_i$ *and* $b_i$ *randomly chosen from any intervals of* $\Re^n$ *and* $\Re$ *respectively, according to any continuous probability distribution, then with probability one, the hidden layer output matrix* ***H*** *of the SLFN is invertible and* $\|H\beta - T\| = 0$.

Lemma 1 illustrates that when the number of neurons $L$ equals to the number of samples $N$, by randomly choosing parameters of neurons, the column vectors of matrix **H** are linearly independent. In fact, the linearly independent property is also correct when $L << N$. In that case, orthonormal neural networks can be constructed by Gram-Schmidt orthonormal transformation.

## B. APPROXIMATION WITH ORTHONORMAL BASIS

Let us assume that training samples are generated with uniform probability distribution and all functions belong to the integrable space $L^2$. As in [152], the inner product of two functions $u(x), v(x)$ can be defined as

$$< u(x), v(x) >= \sum_{i=1}^{N} u(x_i)v(x_i) ,$$

where $N$ is the number of training samples. Here we should note that the above inner product expression is based on statistics, which can be approximated by interpolation. In fact, when the number of training data is large enough, the inner product can be easily deduced by the limitation theory. Thus without loss of generality, we denote the inner product using interpolation hereinafter.

We say that the nonzero vectors $e_1,\cdots,e_n$ are orthogonal if $< e_i, e_j >= 0$, for $i \neq j$ and orthonormal if $< e_i, e_i >= 1$.

For any f and $\{e_k\}_{k=1}^{\infty}$ in Hilbert space *H*, the identity

$$\|f\|^2 = \sum_{k=1}^{\infty} |a_k|^2 \text{ , where } a_k =< f, e_k > \text{ ,}$$

which is called *Parseval identity*, holds if and only if $\{e_k\}_{k=1}^{\infty}$ is also an orthonormal basis ([153], pp.166).

In fact, for any linearly independent sequence $\{g_1(x), g_2(x),\cdots, g_L(x)\}$ in Hilbert space, we can construct an orthonormal basis $\{e_1, e_2,\cdots, e_L\}$ by any orthonormal transformation, such as the Gram-Schmidt process ([153], pp.167-168). Furthermore, we will prove its approximation capability under orthonormal basis as follows:

**Theorem 1** *For any $f \in H$, suppose $e_1, e_2, \cdots$ is an orthonormal basis in H, then*

$$\left\| f - \sum_{j=1}^{L} < f, e_j > e_j \right\| \to 0 \text{, as } L \to \infty .$$

**Proof:** We set $\hat{f} = f - \sum_{j=1}^{\infty} < f, e_j > e_j$ . Since Hilbert space *H* is a complete inner product space ([154], pp.102), we have $\hat{f} \in H$ . Since $e_1, e_2, \cdots$ is an orthonormal basis in *H*, then according to Parseval identity, we have

$$\begin{aligned}
\|\hat{f}\|^2 &= \sum_{k=1}^{\infty} \left| < \hat{f}, e_k > \right|^2 \\
&= \sum_{k=1}^{\infty} \left| f - \sum_{j=1}^{\infty} < f, e_j > e_j, e_k \right|^2 \\
&= \sum_{k=1}^{\infty} \left| < f, e_k > - \left\langle \sum_{j=1}^{\infty} < f, e_j > e_j, e_k \right\rangle \right|^2 \\
&= \sum_{k=1}^{\infty} \left| < f, e_k > - < f, e_k > \right|^2 \\
&= 0
\end{aligned}$$

***Remark:*** In the classic textbooks of functional analysis, we can find similar theorem in one-dimension space. Here we prove that the similar property can be extended into multi-dimension space.

## C. GRAM-SCHMIDT ORTHONORMALIZATION

Based on Theorem 1, our focus is then put on how to construct an orthonormal set of base functions. According to the above analysis, we should find proper parameters such that

$$\beta_1 g_1(x_i) + \cdots + \beta_L g_L(x_i) = f(x_i)\,,\; i = 1, \cdots, N\;,$$

where $t_i = f(x_i)$ .

Multiplying the above equation by $g_j(x_i)$ and adding the corresponding $L$ equations for $i = 1, \cdots, N$ , we have

$$\beta_1 \sum_{i=1}^{N} g_1(x_i) g_j(x_i) + \cdots + \beta_L \sum_{i=1}^{N} g_L(x_i) g_j(x_i) = \sum_{i=1}^{N} f(x_i) g_j(x_i) \text{ , } j = 1, \cdots, L \text{ .}$$

The above equation can be re-written as

$$\beta_1 < g_1(x), g_j(x) > + \cdots + \beta_L < g_L(x), g_j(x) > = < f(x), g_j(x) > \text{ , } j = 1, \cdots, L$$

The above $L$ equations can be rewritten as $\tilde{H}\beta = \tilde{T}$ ,

where $\tilde{T} = (< f(x), g_1(x) >, \cdots, < f(x), g_L(x) >)^T$ and

$$\tilde{H} = \begin{bmatrix} < g_1(x), g_1(x) > & \cdots & < g_L(x), g_1(x) > \\ \vdots & \ddots & \vdots \\ < g_1(x), g_L(x) > & \cdots & < g_L(x), g_L(x) > \end{bmatrix}_{L \times L}$$

We name $\tilde{H}$ as an inner product hidden layer matrix. If $\{g_k(x)\}_{k=1}^{L}$ are orthonormal to each other, the solutions of above equations can be calculated as $\beta_k = < f(x), g_k(x) > = \sum_{i=1}^{N} f(x_i) g_k(x_i)$ . However, as [152] has pointed out, the set of functions $\{g_k(x)\}_{k=1}^{L}$ is not orthonormal to each other. Hence similar to [152], we apply the standard Gram-Schmidt orthonormalization to transform $\{g_1(x), \cdots, g_L(x)\}$ into an orthonormal set of basis functions $\{u_1(x), \cdots, u_L(x)\}$, i.e.,

$$[u_1(x), \cdots, u_L(x)] = [g_1(x), \cdots, g_L(x)] \cdot V$$

where $V$ is an upper triangular matrix whose detailed expression can be found in Eq. 21 in [152].

According to $\{u_i(x)\}_{i=1}^{L}$'s orthonormal property and Theorem 1, such weights $\{\alpha_i\}_{i=1}^{L}$ can ensure the approximation capability of neural networks, i.e.,

$$\left\| f - \sum_{i=1}^{L} \alpha_i u_i(x) \right\| < \varepsilon \text{, where } \alpha_i =< f(x), u_i(x) > .$$

## D. Summary of orthonormal transformation

Gram-Schmidt Orthonormalization as analyzed above requires the column vectors of the original hidden matrix $H$ to be independent of each other. Lemma 1 illustrates that if activation function $g(x)$ is infinitely differentiable, then for almost all the parameters, the column vectors $\left\{ \vec{h}_i \right\}_{i=1}^{L}$ of $H$ are linearly independent of each other. Summarizing the above results, we have the following theorem:

**Theorem 2** *For any bounded, integrable function* $g(a_i, b_i, x) \in L^2$ *, if it is an infinitely differentiable additive function or kernel function, then neural networks by Gram-Schmidt transformation are universal approximation.*

***Remark*:** Theorem 2 shows the approximation capability of orthonormal neural networks, which further explains why the orthonormal neural networks proposed by [152] can achieve good generalization performance. Moreover, Theorem 2 extends the result of [152] from kernel neurons to additive neurons, which as said, is typically of higher classification accuracy.

The learning procedure of the orthonormal neural networks can be summarized in the following steps:

**Algorithm:** given a training set $\{(x_i, t_i) \mid x_i \in \Re^n, t_i \in \Re^m, i = 1, \cdots, N\}$ and $L$ hidden neurons:

step 1) Randomly configure parameters $a_i$ and $b_i$, $i = 1, \cdots, L$.

step 2) Calculate hidden layer output matrix **H**.

step 3) Transform matrix **H** into orthonormal matrix $\tilde{H}$ by any orthonormal transformation.

step 4) Compute the output weights derived by the inner product hidden layer matrix $\tilde{H}$ .

We employ the above Gram-Schmidt orthonormalization algorithm for end-to-end classification of QoS violations. As explained, $x$ is the input vector for classification and in our case, consists of flow parameters such as delay, jitter and video frame rate. $t$ is the target vector which indicates the classification result, which in our case, shows the category a QoS violation belongs to. Other vectors and matrixes are the internal calculating parameters of the neural network whose meanings have been explained in the previous sections.

# APPENDIX B
# AN EXAMPLE ONTOLOGY FOR COMMUNICATION PROTOCOLS

This appendix shows some of the ontology we have developed for QoS middleware profiling.

## A. PROTOCOL.RDF

This file contains the ontology of communication protocols.

```
<!DOCTYPE rdf:RDF>

<rdf:RDF
    xmlns="http://www.comp.nus.edu.sg/dpf/ProtocolSpec#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xml:base="http://www.comp.nus.edu.sg/dpf/ProtocolSpec#">

  <rdfs:Class rdf:ID="Service"/>

  <rdfs:Class rdf:ID="Codec">
    <rdfs:subClassOf rdf:resource="#Service"/>
  </rdfs:Class>

  <rdfs:Class rdf:ID="Media_aware">
    <rdfs:subClassOf rdf:resource="#Service"/>
  </rdfs:Class>

  <rdfs:Class rdf:ID="Transport">
    <rdfs:subClassOf rdf:resource="#Service"/>
  </rdfs:Class>

  <owl:Class rdf:ID="Network">
    <rdfs:subClassOf rdf:resource="#Service"/>
  </owl:Class>

  <owl:Class rdf:ID="JMFCodec">
    <rdfs:subClassOf rdf:resource="#Codec"/>
  </owl:Class>

  <owl:DatatypeProperty rdf:ID="layerName">
    <rdfs:domain rdf:resource="#Service"/>
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#String"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:ID="className">
    <rdfs:domain rdf:resource="#Service"/>
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#String"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:ID="protocolName">
    <rdfs:domain rdf:resource="#Service"/>
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#String"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:ID="supportedInputFormats">
    <rdfs:domain rdf:resource="#Codec"/>
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#String"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:ID="supportedOutputFormats">
    <rdfs:domain rdf:resource="#Codec"/>
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#String"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:ID="upperDependency">
    <rdfs:domain rdf:resource="#Service"/>
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:ID="lowerDependency">
    <rdfs:domain rdf:resource="#Service"/>
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:ID="polarity">
    <rdfs:domain rdf:resource="#Service"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
    <rangeValue>1</rangeValue>
    <rangeValue>0</rangeValue>
    <rangeValue>-1</rangeValue>
  </owl:DatatypeProperty>

</rdf:RDF>
```

## B. INSTANCE.RDF

This file describes the protocol instances available in a real end-host.

```
<!DOCTYPE rdf:RDF>

<rdf:RDF
```

```
  xmlns="http://www.comp.nus.edu.sg/dpf/ProtocolSpec#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://www.comp.nus.edu.sg/dpf/ProtocolSpec#">

<JMFCodec rdf:ID="MPA_RTP_Encoder">
  <layerName>protocol.service.Codec</layerName>
  <protocolName>MPA_RTP</protocolName>
  <className>protocol.plugin.codec.MPA_RTP_Encoder</className>
  <upperDependency>NULL</upperDependency>
  <lowerDependency>protocol.service.Media_aware</lowerDependency>
  <polarity>1</polarity>
</JMFCodec>

<JMFCodec rdf:ID="MPA_RTP_Decoder">
  <layerName>protocol.service.Codec</layerName>
  <protocolName>MPA_RTP</protocolName>
  <className>protocol.plugin.codec.MPA_RTP_Decoder</className>
  <upperDependency>NULL</upperDependency>
  <lowerDependency>protocol.service.Media_aware</lowerDependency>
  <polarity>-1</polarity>
</JMFCodec>

<JMFCodec rdf:ID="G723_RTP_Encoder">
  <layerName>protocol.service.Codec</layerName>
  <protocolName>G723_RTP</protocolName>
  <className>protocol.plugin.codec.G723_RTP_Encoder</className>
  <upperDependency>NULL</upperDependency>
  <lowerDependency>protocol.service.Media_aware</lowerDependency>
  <polarity>1</polarity>
</JMFCodec>

<JMFCodec rdf:ID="G723_RTP_Decoder">
  <layerName>protocol.service.Codec</layerName>
  <protocolName>G723_RTP</protocolName>
  <className>protocol.plugin.codec.G723_RTP_Decoder</className>
  <upperDependency>NULL</upperDependency>
  <lowerDependency>protocol.service.Media_aware</lowerDependency>
  <polarity>-1</polarity>
</JMFCodec>

<JMFCodec rdf:ID="JPEG_RTP_Encoder">
  <layerName>protocol.service.Codec</layerName>
  <protocolName>JPEG_RTP</protocolName>
  <className>protocol.plugin.codec.JPEG_RTP_Encoder</className>
  <upperDependency>NULL</upperDependency>
  <lowerDependency>protocol.service.Media_aware</lowerDependency>
  <polarity>1</polarity>
</JMFCodec>

<JMFCodec rdf:ID="JPEG_RTP_Decoder">
  <layerName>protocol.service.Codec</layerName>
  <protocolName>JPEG_RTP</protocolName>
  <className>protocol.plugin.codec.JPEG_RTP_Decoder</className>
  <upperDependency>NULL</upperDependency>
  <lowerDependency>protocol.service.Media_aware</lowerDependency>
  <polarity>-1</polarity>
</JMFCodec>
```

```
<JMFCodec rdf:ID="H263Encoder">
  <layerName>protocol.service.Codec</layerName>
  <protocolName>H.263</protocolName>
  <className>protocol.plugin.codec.H263Encoder</className>
  <upperDependency>NULL</upperDependency>
  <lowerDependency>protocol.service.Media_aware</lowerDependency>
  <polarity>1</polarity>
</JMFCodec>

<JMFCodec rdf:ID="H263Decoder">
  <layerName>protocol.service.Codec</layerName>
  <protocolName>H.263</protocolName>
  <className>protocol.plugin.codec.H263Decoder</className>
  <upperDependency>NULL</upperDependency>
  <lowerDependency>protocol.service.Media_aware</lowerDependency>
  <polarity>-1</polarity>
</JMFCodec>

<Media_aware rdf:ID="RTPOut">
  <layerName>protocol.service.Media_aware</layerName>
  <protocolName>RTP</protocolName>
  <className>protocol.plugin.media_aware.RTPOut</className>
  <upperDependency>NULL</upperDependency>
  <lowerDependency>protocol.service.Transport</lowerDependency>
  <polarity>1</polarity>
</Media_aware>

<Media_aware rdf:ID="RTPIn">
  <layerName>protocol.service.Media_aware</layerName>
  <protocolName>RTP</protocolName>
  <className>protocol.plugin.media_aware.RTPIn</className>
  <upperDependency>NULL</upperDependency>
  <lowerDependency>protocol.service.Transport</lowerDependency>
  <polarity>-1</polarity>
</Media_aware>

<Transport rdf:ID="UDPOut">
  <layerName>protocol.service.Transport</layerName>
  <protocolName>UDP</protocolName>
  <className>protocol.plugin.transport.UDPOut</className>
  <upperDependency>NULL</upperDependency>
  <lowerDependency>protocol.service.Network</lowerDependency>
  <polarity>1</polarity>
</Transport>

<Transport rdf:ID="UDPIn">
  <layerName>protocol.service.Transport</layerName>
  <protocolName>UDP</protocolName>
  <className>protocol.plugin.transport.UDPIn</className>
  <upperDependency>NULL</upperDependency>
  <lowerDependency>protocol.service.Network</lowerDependency>
  <polarity>-1</polarity>
</Transport>

<Transport rdf:ID="TCPOut">
  <layerName>protocol.service.Transport</layerName>
  <protocolName>TCP</protocolName>
  <className>protocol.plugin.transport.TCPOut</className>
  <upperDependency>NULL</upperDependency>
  <lowerDependency>protocol.service.Network</lowerDependency>
  <polarity>1</polarity>
  </Transport>

  <Transport rdf:ID="TCPIn">
    <layerName>protocol.service.Transport</layerName>
    <protocolName>TCP</protocolName>
    <className>protocol.plugin.transport.TCPIn</className>
    <upperDependency>NULL</upperDependency>
    <lowerDependency>protocol.service.Network</lowerDependency>
    <polarity>-1</polarity>
  </Transport>

  <Network rdf:ID="DummyIPOut">
    <layerName>protocol.service.Network</layerName>
    <protocolName>IP</protocolName>
    <className>protocol.plugin.network.DummyIPOut</className>
    <upperDependency>NULL</upperDependency>
    <lowerDependency>NULL</lowerDependency>
    <polarity>1</polarity>
  </Network>

  <Network rdf:ID="DummyIPIn">
    <layerName>protocol.service.Network</layerName>
    <protocolName>IP</protocolName>
    <className>protocol.plugin.network.DummyIPIn</className>
    <upperDependency>protocol.service.Transport</upperDependency>
    <lowerDependency>NULL</lowerDependency>
    <polarity>-1</polarity>
  </Network>

</rdf:RDF>
```

# ***BIBLIOGRAPHY***

[1] X. Xiao and L. M. Ni, “Internet QoS: A big picture”, *IEEE Network*, 13(2): 8–18, Mar 1999

[2] S. Shenker J. C. Partridge, R. Guerin, “Specification of Guaranteed Quality of Service”, *RFC 2212*, 1997

[3] S. Blake et al., “An Architecture for Differentiated Services,” *RFC 2475*, 1998

[4] E. Crawley et al., “A Framework for QoS-based Routing in the Internet”, *RFC 2386*, 1998

[5] E. Rosen, A. Viswanathan, and R. Callon, “Multiprotocol Label Switching Architecture”, *RFC 3031*, 2001

[6] R. Braden et al., “Resource Reservation Protocol (RSVP) Version 1 Functional Specification”, *RFC 2205*, September 1997

[7] Integrated Services - Cisco Systems, http://www.cisco.com/en/US/products/ps6611/products_ios_protocol_group_home.html

[8] Baochun Li, Klara Nahrstedt, “QualProbes: Middleware QoS Profiling Services for Configuring Adaptive Applications”, *in Proc. IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing*, April 2000

[9] C. Aurrecoechea, A. T. Campbell, L. Hauw, “A survey of QoS architectures”, *Multimedia Systems*, Vol. 6, Issue 3, pp. 138-151, May 1998

[10] Yuming Jiang, Chen-Khong Tham, Chi-Chung Ko, “Challenges and approaches in providing QoS monitoring”, *International Journal of Network Management*, Vol. 10, Issue 6, pp. 323-324, November 2000

[11] Jürg Bolliger, Thomas Gross, “A Framework-Based Approach to the Development of Network-Aware Applications”, *IEEE Transactions on Software Engineering*, Vol. 24, No. 5, pp. 376-390, May 1998

[12] X. Wang and H. Schulzrinne, “Comparison of adaptive Internet multimedia applications”, *IEICE Transactions on Communications*, Vol. E82-B, No. 6, pp. 806-818, June 1999

[13] Hojung Cha, Jaehak Oh and Rhan Ha, “Dynamic Frame Dropping for Bandwidth Control in MPEG Streaming System”, *Multimedia Tools and Applications*, Vol. 19, Number 2, pp. 155-178, February 2003

[14] Viktor S. Wold Eide, Ole-Christoffer Granmo, Frank Eliassen, Jørgen Andreas Michaelsen, “Real-time video content analysis: QoS-aware application composition and parallel processing”, *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP)*, Vol. 2, Issue 2, pp. 149-172, May 2006

[15] Siddhartha Chattopadhyay, Suchendra M. Bhandarkar, Kang Li, “Ligne-claire video encoding for power constrained mobile environments”, *in Proc. the 15th international conference on Multimedia (ACM Multimedia)*, pp. 1036-1045, September 2007

[16] D. P. Anderson, S. Tzou, R. Wahbe, R. Govindan, and M. Andrews, “Support for Continuous Media in the DASH System”, *in Proc. the 10th International Conference on Distributed Computing Systems*, pages 54-61, May 1990

[17] H. Tokuda, T. Nakajima, and P. Rao, “Real-Time Mach: Toward a Predictable Real-Time System”, *in Proc. USENIX Mach Workshop*, pp. 73-82, October 1990

[18] Leslie IM, McAuely D, and Mullender SJ, “Pegasus – Operating Systems support for Distributed Multimedia Systems”, *ACM SIGOPS Operating Systems Review*, Vol. 27, Issue 1, pp. 69 – 78, 1993

[19] N. Klara Nahrstedt, X. Dongyan, W. Duangdao, and L. Baochun, “QoS-aware middleware for ubiquitous and heterogeneous environments”, *IEEE Communications Magazine*, Vol. 39, Issue 11, pp. 140-148, 2001

[20] D. Bauer, B. Stiller, and B. Plattner, “Guaranteed multipoint communication support for multimedia applications”, in *Proc. Broadband European Networks Conference(SYBEN)*, pp. 395-404, May 1998

[21] Fabio Panzieri , Marco Roccetti , Vittorio Ghini, “The implementation of middleware services for QoS-aware distributed multimedia applications”, *in Proc. International Workshop on Multimedia Middleware*, October 05, 2001

[22] B. Li and K. Nahrstedt, “A Control-based Middleware Framework for Quality of Service Adaptations”, *IEEE Journal on Selected Areas in Communications, Special Issue on Service Enabling Platforms*, Vol. 17, No. 9, pp. 1632-1650, September 1999

[23] B. Stiller, C. Class, M. Waldvogel, G. Caronni, and D. Bauer, “A Flexible Middleware for Multimedia Communication: Design, Implementation, and Experience”, *IEEE Journal on Selected Areas in Communications, Special Issue on Middleware*, Vol. 17, No. 9, pp. 1580-1598, September 1999

[24] Mitchell S., H. Naguib, G. Coulouris, and T. Kindberg, “A QoS support framework for dynamically Reconfigurable Multimedia Applications”, *in Proc. Distributed Applications and Interoperable Systems II*, June 1999

[25] Douglas C. Schmidt, Donald F. Box, Tatsuya Suda, “Adaptive: A dynamically assembled protocol transformation, integration and evaluation environment”, *Journal of Concurrency: Practice and Experience*, Volume 5 Issue 4, pp. 269-286, 1993

[26] Christian Tschudin, “Flexible protocol stacks”, *in Proc. ACM SIGCOMM*, pp. 197-205, 1991

[27] G. Tsirtsis, P. Srisuresh, “Network Address Translation - Protocol Translation (NAT-PT)”, *RFC 2766*, February 2000

[28] Dan Chalmers and Morris Sloman, “A survey of Quality of Service in mobile computing environments”, *IEEE Communications Surveys*, Volume 2, Issue 2, April 1999

[29] Guenkova-Luy, T.; Kassler, A.J.; Mandato, D., “End-to-end quality-of-service coordination for mobile multimedia applications”, *IEEE Journal on Selected Areas in Communications*, Vol. 22, Issue 5, pp. 889-903, June 2004

[30] Enthrone project, http://www.ist-enthrone.org/

[31] ITU-T, “ITU-T Rec. G.1000 (11/2001)”, *online document, International Telecommunication Union,* 2001

[32] A. Vogel, B. Kerhervé, G. von Bochmann, J. Gecsei, “Distributed Multimedia and QoS: A Survey”, *IEEE Multimedia*, Vol. 2, Issue 2, pp. 10-18, 1995

[33] T. Bheemarjuna Reddy, I. Karthigeyan, B.S. Manoj and C. Siva Ram Murthy, “Quality of service provisioning in ad hoc wireless networks: a survey of issues and solutions”, *Ad Hoc Networks*, Vol. 4, Issue 1, pp. 83-124, January 2006

[34] Y. Bernet et al., “A Framework for Integrated Services Operation over DiffServ Networks”, *RFC 2998*, November 2000

[35] Jaudelice C. de Oliveira, Caterina Scoglio, Ian F. Akyildiz and George Uhl, "New preemption policies for DiffServ-aware traffic engineering to minimize rerouting in MPLS networks", *IEEE/ACM Transactions on Networking (TON)*, Vol. 12 Issue 4, pp. 733-745, August 2004

[36] Mohsin Iftikhar, Bjorn Landfeldt and Mine Caglar, "Traffic engineering and QoS control between wireless DiffServ domains using PQ and LLQ", *in Proc. the 5th ACM International Workshop on Mobility Management and Wireless Access (MobiWac)*, pp, 120-129, October 2007

[37] Ikjun Yeom, A. L. Narasimha Reddy, "Modeling TCP behavior in a differentiated services network", *IEEE/ACM Transactions on Networking (TON)*, Volume 9, Issue 1, pp. 31-46, February 2001

[38] Kartik Gopalan, Lan Huang, Gang Peng, Tzi-Cker Chiueh, Yow-Jian Lin, "Statistical admission control using delay distribution measurements", *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP)*, Vol. 2 Issue 4, pp. 258-281, November 2006

[39] Dong Lin, Robert Morris, "Dynamics of random early detection", *in Proc. ACM SIGCOMM*, pp. 127-137, October 1997

[40] "Hard Real-Time with Venturcom RTX on Microsoft Windows XP and Windows XP Embedded", http://msdn.microsoft.com/en-us/library/ms838583.aspx

[41] A. Kassler, B. Reiterer, J. Kaiser, "A Framework for scheduling soft real-time multimedia applications", *in Proc. Internet and Multimedia Systems and Applications (IMSA)*, Honolulu, USA, August 2003

[42] "Cisco Completes DiffServ Solution with Value-added Tools for End-to-End QoS", http://newsroom.cisco.com/dlls/corp_041001.html

[43] B. Stiller, C. Class, M. Waldvogel, G. Caronni, D. Bauer and B. Plattner, "A Flexible Middleware for Multimedia Communication: Design, Implementation, and Experience", *IEEE Journal on Selected Areas in Communications: Special Issue on Middleware*, Vol. 17, No. 9, pp. 1614-1631, 1999

[44] Gerard Parr, Kevin Curran, "A Paradigm Shift in the Distribution of Multimedia", *Communications of the ACM*, Vol. 43, No. 6, pp. 103-109, June 2000

[45] Kevin Curran, Gerard Parr, "A middleware architecture for streaming media over IP networks to mobile devices", *in Proc. IEEE Wireless Communications and Networking Conference (WCNC)*, 2003

[46] Liming An, Hung Keng Pung, Lifeng Zhou, "Design and Implementation of a Dynamic Protocol Framework", *Computer Communications*, Vol. 29, Issue 9, pp. 1309-1315, May 2006

[47] K. Nahrstedt, D. Wichadakul, and D. Xu, "Distributed QoS Compilation and Runtime Instantiation", *in Proc. the 8th IEEE/IFIP International Workshop on Quality of Service (IWQoS)*, pp. 198–207, June 2000

[48] Denise J. Ecklund,Vera Goebel,Thomas Plagemann, Earl F. Ecklund, Jr., "Dynamic end-to-end QoS management middleware for distributed multimedia systems", *Multimedia Systems*, Vol. 8 , Issue 5, pp. 431-442, December 2002

[49] Pryce N. G., "Component Interaction in Distributed Systems", *PhD Thesis*, University of London and Diploma of the Imperial College of Science, Technology and Medicine, January 2000

[50] Santa Fe, New Mexico, "An evaluation of Qinna, a component-based QoS architecture for embedded systems", *in Proc. ACM symposium on Applied computing*, pp. 998-1002, 2005

[51] Pravin Pawar, Katarzyna Wac, Bert-Jan van Beijnum, Pierre Maret, Aart van Halteren, Hermie Hermens, "Context-aware middleware architecture for vertical handover support to multi-homed nomadic mobile services", *in Proc. ACM Symposium on Applied Computing*, pp. 481-488, March 2008

[52] Heinzelman W.B., Murphy A.L., Carvalho, H.S., Perillo, M.A., "Middleware to support sensor network applications", *IEEE Network*, Vol. 18, Issue 1, pp. 6-14, January-February 2004

[53] Tatsuo Nakajima, "Experiences with building middleware for audio and visual networked home appliances on commodity software", *in Proc. the tenth ACM international conference on Multimedia*, December 01-06, pp. 611-620, 2002

[54] Internet2 QoS working group, "Network QoS needs of advanced Internet applications, a survey", *online document*, http://qos.internet2.edu/wg/apps-/fellowship/Docs/Internet2AppsQoSNeeds.pdf

[55] Aravind, R., Civanlar, M. R., Reibman, A.R., "Packet loss resilience of MPEG-2 scalable video coding algorithms", *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 6, Issue: 5, pp. 426-435, Oct 1996

[56] Jae-Hyun Kim, Hyun-Jin Lee, Sung-Min Oh and Sung-Hyun Cho, "Performance modeling and evaluation of data/voice services in wireless networks", *Wireless Networks*, Volume 14, Issue 2, pp. 233-246, March 2008

[57] Microsoft Corporation, "Key Concepts in Windows Media Technologies", http://msdn.microsoft.com

[58] Sun Microsystems, Inc., "Java Media Framework API", http://www.javasoft.com/jmf

[59] H. Schulzrinne, S. Casner, etc., "RTP: A Transport Protocol for Real-Time Applications", *RFC 3550*, 2003

[60] H. Schulzrinne, Columbia U., etc., "Real Time Streaming Protocol (RTSP)", *RFC 2326*, 1998

[61] Lai-U Choi, Wolfgang Kellerer, and Eckehard Steinbach, "Cross-layer optimization for wireless multi-user video streaming", *in Proc. IEEE International Conference on Image Processing (ICIP)*, 2004

[62] Q. Zhang, W. Zhu, and Y-Q. Zhang, "A Cross-layer QoS-supporting framework for multimedia delivery over wireless Internet", i*n Proc. International Packetvideo workshop*, 2002

[63] Toufik Ahmed, Ahmed Mehaoua, etc., "Adaptive Packet Video Streaming Over IP Networks: A Cross-Layer Approach", *IEEE Journal on Selected Areas in Communications*, Vol. 23, No. 2, pp. 385-401, February 2005

[64] Wanghong Yuana, etc., "Design and Evaluation of a Cross-Layer Adaptation Framework for Mobile Multimedia Systems", *in Proc. SPIE/ACM Multimedia Computing and Networking Conference*, 2003

[65] Arif Ghafoor, "Distributed multimedia information systems: an end-to-end perspective", *Multimedia Tools and Applications*, Vol. 33, No. 1, pp. 31-56, April 2007

[66] A. Campbell, G.Coulson, and D. Hutchinson, "A Quality of Service architecture", *Computer Communications Review*, Vol. 20, No. 4, pp. 200-208, September 1990

[67] Klara Nahrstedt, "An architecture for end-to-end quality of service provision and its experimental validation", *PhD Thesis*, University of Pennsylvania, 1996

[68] Abdelhakim Hafid and Gregor v. Bochmann, "Quality-of-service adaptation in distributed multimedia applications", *Multimedia Systems*, Vol. 6, Issue 5, pp. 299-315, September 1998

[69] Daniel Won-Kyu Hong, Choong Seon Hong, “A QoS management framework for distributed multimedia systems”, *International Journal of Network Management*, Volume 13, Issue 2, pp. 115-127, March/April 2003

[70] Abdelhakim Hafid, Gregor V., Bochmann, “An Approach to Quality of Service Management in Distributed Multimedia Application: Design and an Implementation”, *Multimedia Tools and Applications*, Vol. 9, Issue 2, pp. 167-191, September 1999

[71] Manish, Mahajan and Parashar, “Managing QoS for Multimedia Applications in the Differentiated Services Environment”, *Journal of Network and Systems Management*, Vol. 11 , Issue 4, pp. 469-498, December 2003

[72] Zahi Jarir and Mohammed Erradi, “A meta-level architecture for QoS awareness in a mobile environment”, *in Proc. 8th international conference on New technologies in distributed systems*, 2008

[73] Wei Zha, Rose Qingyang Hu, Yi Qian, Yu Cheng, “An adaptive MAC scheme to achieve high channel throughput and QoS differentiation in a heterogeneous WLAN”, *in Proc. the 3rd international conference on Quality of service in heterogeneous wired/wireless networks (QShine)*, 2006

[74] Aimin Sang, Xiaodong Wang, etc., “Fairness and load balancing: Coordinated load balancing, handoff/cell-site selection, and scheduling in multi-cell packet data systems”, *Wireless Networks,* pp. 103-120, 2004

[75] Weiguang Shi, M. H. MacGregor, Pawel Gburzynski, “Load balancing for parallel forwarding”, *IEEE/ACM Transactions on Networking*, Vol. 13, Issue 4, pp. 790-801, August 2005

[76] F. Dabek, N. Zeldovich, M. F. Kaashoek, D. Mazières, and R.Morris, “Event-driven programming for robust software”, *in Proc. the 10th ACM SIGOPS European Workshop*, pp. 186-189, September 2002

[77] D. D. Clark, C. Partridge, J. C. Ramming, and J. T. Wroclawski, “A knowledge plane for the Internet”, *in Proc. ACM SIGCOMM*, pp. 3-10, August 2003

[78] A. De Paola, S. Fiduccia, S. Gaglio, etc., “Rule-Based Reasoning for Network Management”, *in Proc. International Workshop on Computer Architecture for Machine Perception (CAMP)*, pp. 25-30, 2005

[79] M. Wawrzoniak, L. L. Peterson, and T. Roscoe, “Sophia: an information plane for networked systems”, *ACM Computer Communication Review*, Vol. 34, No. 1, pp. 15-20, January 2004

[80] Michael J. Katchabaw, Hanan L. Lutfiyya, Michael A. Bauer, “Driving resource management with application-level quality of service specifications”, *in Proc. the first International Conference on Information and Computation Economies*, pp. 83-91, 1998

[81] Zoubir Mammeri, “Towards a Formal Model for QoS Specification and Handling in Networks”, *in Proc. International Workshop on Quality of Service (IWQoS)*, pp. 148-152, 2004

[82] Chen Zhou, Liang-Tien Chia, and Bu-Sung Lee, “QoS Measurement Issues with DAML-QoS Ontology”, *in Proc. IEEE International Conference on e-Business Engineering (ICEBE)*, pp. 395-403, 2005

[83] Glen Dobson, Russell Lock, Ian Sommerville, “QoSOnt: an Ontology for QoS in Service-Centric Systems”, *in Proc. UK e-Science AHM*, pp. 80-87, 2005

[84] E. Michael Maximilien, Munindar P. Singh, "A Framework and Ontology for Dynamic Web Services Selection", *IEEE Internet Computing*, 8(5):84-93, September-October 2004

[85] Dan Brickley, R.V. Guha, "RDF Vocabulary Description Language 1.0: RDF Schema", *World Wide Web Consortium*, January 2003

[86] Deborah L. McGuinness, Frank van Harmelen, "OWL Web Ontology Language Overview", *World Wide Web Consortium*, Feburary 2004

[87] Paolo Bellavista, Antonio Corradi, Rebecca Montanari, Cesare Stefanelli, "A Mobile Computing Middleware for Location and Context-aware Internet Data Services", *ACM Transactions on Internet Technology*, Volume 6 Issue 4, pp. 356-380, November 2006

[88] Jingwen Jin, Klara Nahrstedt, "QoS Specification Languages for Distributed Multimedia Applications: A Survey and Taxonomy", *IEEE Multimedia Magazine*, Vol. 11, Issue 3, pp. 74-87, July-September 2004

[89] Richard Staehli, Frank Eliassen, Sten Amundsen, "Designing adaptive middleware for reuse", *in Proc. the 3rd ACM workshop on Adaptive and reflective middleware*, pp. 189-194, October 2004

[90] Ernesto Exposito, Mathieu Gineste, etc., "XQOS: a Quality of Service specification language", *in Proc. International Conference on WWW/Internet*, 2002

[91] M.A., de Miguel, "QoS modeling language for high quality systems", *in Proc. International Workshop on Object-Oriented Real-Time Dependable Systems*, pp. 210-216, 2003

[92] A.L Murphy, G. P. Picco, G.-C Roman, "Lime: a middleware for physical and logical mobility", *in Proc. IEEE International Conference on Distributed Computing Systems*, pp. 524-533, 2001

[93] Sabata, B., Chatterjee, S. , etc., "Taxonomy for QoS specifications", *in Proc. International Workshop on Object-Oriented Real-Time Dependable Systems*, pp. 100-107, 1997

[94] G. Xiaohui, N. Klara, Y. Wanghong, W. Duangdao, X. Dongyan, "An XML-based QoS enabling language for the Web", *Journal of Visual Language and Computing, Special Issue on Multimedia Languages for the Web*, 13(1): 61-95, February 2002

[95] S. Frlund, J. Koistinen, "Quality-of-Service specification in distributed object systems design", *in Proc. USENIX Conference on Object-Oriented Technologies and Systems*, pp. 1-1, April 1998

[96] G. Ghinea, J.P. Thomas, "QoS impact on user perception and understanding", *in Proc. ACM Multimedia*, pp. 49-54, 1998

[97] Oliver T. W. Yu, "End-to-end adaptive QoS provisioning over GPRS wireless mobile network", *Mobile Networks and Applications*, Vol. 8, Issue 3, pp. 255-267, June 2003

[98] Indulska, J., Balasubramaniam, S., "Context-aware vertical handovers between WLAN and 3G networks", *in Proc. IEEE Vehicular Technology Conference*, pp. 3019-3023, 2004

[99] Yuan Feng, "iSSE: an Intelligent search engine for Web service", *MS.C. thesis*, School of Computing, National University of Singapore, 2005

[100] Vangelis Gazis, etc, “Metadata Design for Reconfigurable Protocol Stacks in Systems Beyond 3G”, *Wireless Personal Communications*, Volume 36 , Issue 1, pp. 1-28, January 2006

[101] Pearson Malcom E, “Dynamic layered protocol stack”, *US patent US5903754*, 1999

[102] I. Sora, etc. “Policies for dynamic stack composition”, *Technical Report*, Department of Computer Science, Leuven, Belgium, 2001

[103] M. Handley et al., “SIP: Session Initiation Protocol”, *RFC2543*, Mar. 1999

[104] Wuttipong Kumwilaisak, Y. Thomas Hou, etc., “A Cross-layer Quality-of-Service mapping architecture for video delivery in wireless networks”, *IEEE Journal on Selected Areas in Communications*, Vol. 21, No. 10, pp. 1685-1698, December 2003

[105] Sun Microsystems, Inc., “Java Remote Method Invocation”, http://java.sun.com/products/jdk/rmi/

[106] C.L Forgy, “RETE: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem”, *Artificial Intelligence*, 1982

[107] L. Combaz, J.-C. Fernandez, J. Sifakis, and L. Strus, “Symbolic quality control for multimedia applications”, *Real-Time Systems*, Vol. 40, pp. 1-43, 2008

[108] C. W. A. Al-Ta’ani, A.; Begall, “Trouble shooting in distributed service centers”, *in Proc. ICNS Third International Conference on Networking and Services*, pp. 6-6, 2007

[109] G. Cormode and S. Muthukrishnan, “What’s new: finding significant differences in network data streams”, *IEEE/ACM Transactions on Networking*, Vol. 13, pp. 1219-1232, 2005

[110] L. Abeni, T. Cucinotta, G. Lipari, L. Marzario, and L. Palopoli, "QoS management through adaptive reservations", *Real-Time Systems*, Vol. 29, pp. 131-155, March 2005

[111] L. Lymberopoulos, E. Lupu, and M. Sloman, "An adaptive policy based framework for network services management", *Journal of Network and Systems Management*, Vol. 11, pp. 277-303, 2003

[112] Y. Huang, N. Feamster, A. Lakhina, and J. J. Xu, "Diagnosing network disruptions with network-wide analysis", *in Proc. ACM SIGMETRICS*, pp. 61-72, June 2007

[113] V. Bharghavan, K.-W. Lee, S. Lu, S. Ha, J.-R. Li, and D. Dwyer, "The timely adaptive resource management architecture", *IEEE Personal Communications*, Vol. 5, No. 4, pp. 20-31, August 1998

[114] D. Gmach, S. Krompass, A. Scholz, M. Wimmer, and A. Kemper, "Adaptive quality of service management for enterprise services", *ACM Transactions on the Web*, Vol. 2, no. 1, pp. 1-46, February 2008

[115] G. Molenkamp and M. Katchabaw, "Managing soft QoS requirements in distributed systems", *in Proc. International Workshops on Parallel Processing*, pp. 185-201, 2000

[116] A. Ghafoor, "Distributed multimedia information systems: an end-to-end perspective", *Multimedia Tools and Applications*, Vol. 33, pp. 31-56, 2007

[117] Nishanth Shankaran, Xenofon Koutsoukos, Douglas C. Schmidt, Aniruddha Gokhale, "Evaluating adaptive resource management for distributed real-time embedded systems", *in Proc. the 4th workshop on Reflective and adaptive middleware systems*, November 2005

[118] A. Hafid and G. v. Bochmann, "Quality-of-service adaptation in distributed multimedia applications", *Multimedia Systems,* Vol. 6, pp. 299-315, 1998

[119] R. M. Bahati, M. A. Bauer, and E. M. Vieira, "Policy-driven autonomic management of multi-component systems", *in Proc. conference of the center for advanced studies on Collaborative Research*, pp. 137-151, 2007

[120] R. Iyer, L. Zhao, F. Guo, and etc., "QoS policies and architecture for cache/memory in CMP platforms", *in Proc. ACM SIGMETRICS,* pp. 25-36, June 2007

[121] J. van der Merwe, S. Sen, C. Kalmanek, "Streaming Video Traffic: Characterization and Network Impact", *in Proc. the 7th International Workshop on Web Content Caching and Distribution*, 2002

[122] O. S. e. M. Roughan, S. Sen, "Class-of-service mapping for QoS: a statistical signature-based approach to IP traffic classification", *in Proc. the 4th ACM SIGCOMM conference on Internet measurement*, pp. 135-148, 2004

[123] M. Nazeeruddin, M. Mohandes, and H. Cam, "ATM QoS prediction using neural-networks", *in Proc. the 6th International Conference on Neural Information Processing,* pp. 532-537, 1999

[124] S. Khoukhi, L.; Cherkaoui, "A quality of service approach based on neural networks for mobile ad hoc networks", *in Proc. the 2nd IFIP International Conference on Wireless and Optical Communications Networks*, pp. 295-301, 2005

[125] A. Nogueira, P. Salvador, and R. Valadas, "Predicting the quality of service of wireless LANs using neural networks", *in Proc. ACM International Symposium on Modeling Analysis and Simulation of Wireless and Mobile Systems*, pp. 52-60, 2007

[126] S. Cen, P. C. Cosman, and G. M. Voelker, “End-to-end differentiation of congestion and wireless losses”, *IEEE/ACM Transactions on Networking*, Vol. 11, No. 5, pp. 703-717, 2003

[127] S. Taylor and H. Lutfiyya, “Predicting violations of QoS requirements in distributed systems”, *in Proc. of IFIP/IEEE International Conference on Management of Multimedia Networks and Services*, pp. 355-367, 2003

[128] S. R. Gulliver and G. Ghinea, “Defining user perception of distributed multimedia quality”, *ACM Transactions on Multimedia Computing, Communications, and Applications*, Vol. 2, No. 4, pp. 241-257, November 2006

[129] N. Hohn and D. Veitch, “Inverting sampled traffic”, *IEEE/ACM Transactions on Networking*, Vol. 14, No. 1, pp. 68-80, 2006

[130] M. Leshno, V.Y.Lin, A. Pinkus, and S. Schocken, “Multilayer feedforward networks with a nonpolynomial activation function can approximate any function”, *Neural Networks*, Vol. 6, pp. 861-867, 1993

[131] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, “Real-time learning capability of neural networks”, *IEEE Transactions on Neural Networks*, Vol. 17, No. 4, pp. 863-878, 2006

[132] J. Platt, “A resource-allocating network for function interpolation”, *Neural Computation*, 3: 213-225, 1991

[133] Visakan Kadirkamanathan and Mahesan Niranjan, “A Function Estimation Approach to Sequential Learning with Neural Networks”, *Neural Computation*, Vol. 5, pp. 954-975, 1993

[134] Lu Yingwei and P. Saratchandran and Narasimhan Sundararajan, “Performance Evaluation of A Sequental Minimal Radial Basis Function (RBF) Neural

Networks Learning Algorithm", *IEEE Transactions on Neural Networks*, Vol. 9, No. 2, pp. 308-318, 1998

[135] M. T. Hagan and M. B. Menhaj, "Training feedforward networks with the marquardt algorithm", *IEEE Transactions on Neural Networks*, Vol. 5, No. 6, pp. 989-993, 1994

[136] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. MÄuller, "Efficient BackProp", *Lecture Notes in Computer Science*, Vol. 1524, pp. 9-50, 1998

[137] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: Theory and applications", *Neurocomputing*, Vol. 70, pp. 489-501, 2006

[138] G.-B. Huang, L. Chen, and C.-K. Siew, "Universal approximation using incremental constructive feedforward networks with random hidden nodes", *IEEE Transactions On Neural Networks*, Vol. 17, No. 4, pp. 879-892, 2006

[139] David D. Clark, etc., "Tussle in cyberspace: defining tomorrow's Internet", *in Proc. ACM SIGCOMM*, pp. 462-475, June 2002

[140] Sugih Jamin, Scott J. Shenker, Peter B. Danzig, "Comparison of Measurement-based Admission Control Algorithms for Controlled-Load Service", *in Proc. IEEE Infocom*, pp. 973-980, 1997

[141] Aleksandar Kuzmanovic, Edward Knightly, "Measurement-Based Characterization and Classification of QoS-Enhanced Systems", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 14 , Issue 7, pp. 671-685, July 2003

[142] Lee Breslau, Edward W. Knightly, Scott Shenker, Ion Stoica, Hui Zhang, "Endpoint admission control: architectural issues and performance", *in Proc. ACM SIGCOMM*, pp. 57-69, August 2000

[143] Seung Yeob Nam, Sunggon Kim, Dan Keun Sung, "Measurement-based admission control at edge routers", *IEEE/ACM Transactions on Networking* (TON), Vol. 16, Issue 2, pp. 410-423, April 2008

[144] Thomas M. Chen, "Network Traffic Modeling", *Handbook of Computer Networks*, Hossein Bidgoli (ed.), Wiley, 2007

[145] Dimitri Bertsekas and Robert Gallager, "Data Networks", *Prentice-Hall, Inc. ISBN:0-13-200916-1*, 1992

[146] The Network Simulator - ns-2, http://isi.edu/nsnam/ns/

[147] WinPcap, http://www.winpcap.org

[148] Jena 2, http://www.hpl.hp.com/semweb/jena2.htm

[149] PlanetLab, http://www.planet-lab.org/

[150] SPARQL Query Language for RDF, http://www.w3.org/TR/rdf-sparql-query/, *W3C Working Draft*, 2006

[151] Anukool Lakhina, Mark Crovella and Christophe Diot, "Mining anomalies using traffic feature distributions", *in Proc. ACM SIGCOMM*, pp. 217-228, 2005

[152] W. Kaminski and P. Strumillo, "Kernel orthonormalization in radial basis function neural networks", *IEEE Transactions on Neural Networks*, Vol. 8, No. 5, pp. 1177-1183, 1997

[153] E. M. Stein and R. Shakarchi, "Real Analysis: Measure Theory, Integration, and Hilbert Spaces", *Princeton University Press*, 2005

[154] J. M. Ortega, "Matrix Theory", *Plenum Press*, New York, 1987