# Characterizing QoS-Awareness in Multimedia Operating Systems

Biswajit Ghose, Vikas Jain, Vandana Gopal
*{bghose, vjain, gvandana}@cs.ucsd.edu*

## *Abstract*

*Supporting Quality of Service (QoS) requirements for multimedia applications places significant demands on Operating Systems as a resource manager for managing end-system resources. We approach the key issues involved in various sub-systems (CPU scheduler, Memory, Network Interface, I/O) towards meeting these QoS requirements. We do so by characterizing a framework for a QoS-Aware OS that includes policies for static and dynamic resource management. We also study and analyze some existing implementations (Chorus, WashU, Omega, HeiRAT) and evaluate them against our framework.*

## 1. Introduction

Faster CPUs, cheaper memories and gigabytes of bandwidth have accelerated the exploration of domains that can leverage on them. Multimedia (MM) is one such real time (RT) application that is load intensive on all parts of the system, typically – CPU, memory, disk I/0, file system, and network. Thus comes the need for a multimedia specific operating system that can handle intensive demands on all of its resources. There are certain characteristics of multimedia applications that have to be considered while designing such an operating system –

- Multimedia applications are time critical with temporal relationships between media (e.g. Movie on demand needs frames to arrive at precise times with sync between audio, video, and perhaps, text).
- Quality of Service (QoS): different MM applications have different QoS requirements and have to be handled accordingly. This poses some demands on the system, e.g. scheduling policies need to emphasize on meeting deadlines, memory management needs to ensure pages are pinned down with bounded access etc.

A MM OS that supports QoS must have the entire system and each of its components acting in *awareness* of the QoS guarantees given.

In this paper, we have attempted to provide the framework for a QoS – Aware Multimedia Operating System, giving some salient requirements for each of the OS subsystems. In this light, we have surveyed some existing QoS supporting MM OS – Chorus, Omega, WashU, and HeiRAT, and presented our critique of them against the proposed framework.

The rest of the paper is structured as follows. Section 2 gives a background on a QoS end-to-end system describing the application classes. Section 3 describes the Resource Management mechanisms. Section 4 characterizes a framework for QoS-aware OS and we follow it up with a survey of some existing implementations in Section 5. Section 6 summarizes this survey, presents some observations and suggestions for future research and we conclude in Section 7.

## 2. Background on QoS

Quality of Service (QoS) is the collective measure of the level of service requested by the user. It can be characterized by several basic performance criteria, e.g error-rates, throughput, delays, jitter etc. For truly meeting quality of service guarantees in a distributed environment, the QoS guarantees must be made *end-to-end*. This in turn means that both the network and the end-system have to contribute towards achieving this end-to-end QoS. Fig [1] illustrates a basic diagram of the entities involved in providing this end-to-end QoS. In this paper, we focus mainly on end-system aspects.

Majority of multimedia applications, by virtue of their inherent nature, can be broadly categorized as isochronous[4] and non-isochronous. Isochronous applications are typically the continuous media applications that generate or consume data at a fixed rate. Non-isochronous applications are characterized by bulk data transfer.

The degree of commitment with which the QoS requirements are met depend on the cost the user is willing to pay for the service. Based on this, the applications are classified as [2] –

a) **Guaranteed** – all deadlines are guaranteed to be met all the time. This application gets highest priority and will need resources to be reserved considering the worst case situation.
b) **Statistical** – deadlines are guaranteed to be met with a certain probability (e.g. a service that guarantees that 90% of the deadlines will be met over an interval). The statistical behavior of this class needs to be monitored and maintained.
c) **Best Effort** – no guarantees are given for meeting deadlines. Deadlines are met on a best-effort basis with the resources leftover from those reserved for guaranteed services. These applications can be pre-empted by higher priority classes, and no resources are reserved.

Consider, for example, the remote playback of a video sequence. Fig [2] illustrates the basic activity diagram for transporting this video stream from the media server over the network to the playback application. As illustrated, the operating system for supporting multimedia applications, in its role as a resource manager, has to assure that the QoS guarantees are met across all the activities spanning its various subsystems. We will only limit ourselves to the QoS issues within the end-system as the network aspect is beyond the scope of this paper.

## 3. Key QoS Mechanism : Resource Management

The primary concern in a QoS-aware multimedia OS is the optimal utilization of resources to provide a maximum number of applications with their requested level of service. *Resource Management* is thus a key mechanism for achieving the desired behaviour [1].

Resource management has two components: *static* and *dynamic*. Static resource management deals with session establishment and end-to-end QoS guarantees (also called QoS provisioning). Dynamic resource management deals with maintaining QoS guarantees through the length of the sessions, and having other control mechanisms like policing.

### *3.1. Static Resource Management*

Static resource management is to be done at session establishment for every application. This is done in three phases –

a) *QoS Translation* – QoS translation involves mapping the application's QoS requirements into the system's resource requirements and comprises of :

- *Distribution* – The responsibility of meeting QoS requirements of the user are distributed among the resources e.g. application end-to-end delay of 2 ms may be split among CPU delay, network delay and disk access time. Resources cooperate and compensate towards meeting QoS requirements.
- *Translation* – The requirements distributed to each resource is translated into the corresponding resource parameters e.g. thread scheduling period is determined by the size of data unit and the rate at which the cells are paced out on a network connection. [4].

b) *Admission Control* – Depending on the current utilization of resources and the QoS requirements of requesting application, the resource manager decides whether to admit the requesting application or not. For the end system, this would imply admission control tests for all resource modules through which the flow traverses (CPU, memory, network interface, I/O).

c) *Resource Reservation* – This provides end-to-end resource reservations according to the QoS requirements of the application. More specifically for the end system, it implies a resource reservation for the resources like memory buffers, CPU thread quanta [3], network bandwidth.

### *3.2. Dynamic Resource Management*

Once the session is established and guarantees given, it has to be ensured that the requirements are met continuously. The following phases are done continuously at run-time for each application:

a) *Scheduling* – The admitted applications need to be scheduled using real time scheduling mechanisms to satisfy their QoS requirements for the length of the session.

b) *Maintenance* – Each application is monitored to see how much of resources it is utilizing. For guaranteed applications – 100% requests are being met, statistical applications – their statistical guarantees are being met, and for best effort – as best as available. In case they are not being met due to degradation of resource or overload scenario, suitable actions are taken, like re-negotiation.

c) *Policing* – Each application is also monitored to check if it utilizing no more than the stated requirements. Suitable penalty measures like session termination or re-negotiation can be taken for misbehaving applications.

We can summarize the above discussion by means of a flow chart as shown in Fig [3].

## 4. Framework for QoS-Aware OS

In a system supporting QoS, each system resource needs to be individually *QoS-Aware*. In this section we attempt to

characterize resource management policies of the following resources – CPU Scheduling, Memory Management, Network Interface, and I/O Subsystem.

### 4.1. CPU Scheduling

In this section, we attempt to derive some requirements for CPU scheduling based on the generalized QoS framework. We think there are some other criteria like fairness criteria that come out of the implicit assumption that we shall be supporting varying degrees of service commitments for application classes like guaranteed, statistical and best effort, in addition to non-multimedia applications.

- **Admission Criteria:** This requirement comes directly from the admission control required for end-to-end QoS provisioning. In CPU scheduling terms it means that admission of a new real-time application should not infringe upon the QoS guarantees given to currently running applications. If so, necessary steps need to be taken like re-negotiation or rejecting the application.
- **Throughput Criteria:** The scheduling policy should be able to schedule as many threads as possible. Specifically, there shouldn't be a thread that doesn't get scheduled even when there is CPU bandwidth available simply because of conservative reservation policies.
- **Real-time Scheduling Guarantees:** This is the most basic assumption for designing a multimedia OS, and a criterion that must be satisfied by all resource modules. More specifically, for CPU scheduling, it implies design of scheduling algorithms (like Rate Monotonic (RM), or Earliest Deadline First (EDF)) that satisfy real-time constraints in terms of ensuring guaranteed scheduling for isochronous threads within their jitter bounds and for non-isochronous threads by their deadlines [3].
- **Fairness Criteria:** It should be possible to schedule all types of threads that are competing for CPU. This implies that the lower priority non-guaranteed applications like Best Effort should not be starved out of CPU by higher priority tasks corresponding to Guaranteed services. As an extension, it should also be possible to schedule aperiodic requests from non-multimedia applications.
- **Maintenance and Policing Criteria:** This requirement comes from the dynamic resource management criteria for end-to-end QoS control. Policing requires ensuring deadline-violating tasks do not infringe upon the QoS guarantees of other tasks competing for CPU resources. Mechanisms like software watchdog that suspends a thread on deadline violations, are means of ensuring service guarantees. Maintenance criteria in the context of CPU implies setting up re-negotiations or dropping further requests (admission control) in case of CPU overload condition.
- **Benchmarking Criteria:** The performance of the multimedia OS while exclusively running non-multimedia applications should not significantly degrade as compared to running these same applications on a non-multimedia OS. This is a general requirement that should be satisfied by all the subsystems of the MM OS.

### 4.2. Memory Management

The following are the requirements of Memory Management in order to support QoS guarantees:

- **Admission Criteria**: New applications can be admitted only when their memory buffer requirements, along with the current buffer allocations of other processes, does not exceed a threshold of the total available memory.
- **Real-time Guarantees:** Some mechanisms to ensure real-time guarantees include –
  - **Bounded Access Latency**: Time critical media applications need memory access time to be minimal. With virtual memory, it is important to have paging mechanisms that have an acceptable upper bound on access latency.
  - **Minimal Copying Semantics**: Mechanisms like Shared Memory may be used minimize copying overheads among co-operating threads.
- **Fairness Criteria**: This ensures the minimal availability of memory buffers for all multimedia application classes. This could be ensured by possibly dimensioning the buffers (during system startup) for all application classes.
- **Maintenance and Policing Criteria**: Maintenance criteria requires setting up re-negotiations or dropping further requests (admission control) in case of a buffer overload scenario. Policing criteria may require the offending application (which have taken more than their requested share of buffers) be notified for further re-negotiation, or in the extreme case, terminate the session (after releasing all its buffers).

### 4.3. Network Interface

The following are the requirements on the Network Interface module in order to support QoS guarantees:

- **Admission Criteria:** This requirement comes directly from the admission control required for end-to-end QoS provisioning. It must ensure that the requested bandwidth plus the current allocated bandwidth does not exceed a threshold of the total bandwidth.
- **Real-time Guarantees:** For guaranteeing end-to-end delays, it must be ensured at the network interface that the network interface scheduling delays are bounded and that enough buffer provisioning is done at the network interface. Also mechanisms like Forward Error Correction, or re-transmissions upon lost packets may not be suitable for applications requiring hard delay bounds.
- **Fairness Criteria:** This requirement comes out of the implicit assumption that we shall be supporting various types of multimedia applications as well as non-multimedia applications. Provisioning may be done to ensure that all these types of applications get a fair share of network bandwidth.
- **Maintenance and Policing Criteria:** This requirement comes from the dynamic resource management criteria for end-to-end QoS control. Policing requires ensuring applications do not take up more than the network bandwidth that has been guaranteed by QoS negotiation during session setup.

### 4.4. I/O Systems

The following are the static and dynamic resource management aspects as applicable to the I/O subsystem:

- **Admission Criteria:** The effective disk transfer bandwidth is reduced due to seek and latency overheads, which are a function of the disk scheduling algorithm and the disk request size. The admission criterion ensures that the sum total of the data rates of all streams, including the new one, do not exceed the effective disk transfer bandwidth.
- **Resource reservation:** In addition to the reservation of disk transfer bandwidth, each real-time stream requires at a minimum a buffer for the consuming process and a buffer for the producing process, thus leading to memory reservation requirement of 2S for each stream, S being the request size. This amount of memory also needs to be reserved for each admitted stream.
- **Real-time Scheduling guarantees :** The requirements on the disk-scheduling algorithm are the following:
  - To be able to schedule the disk accesses for all admitted streams so as to meet their data rate guarantees.
  - The response time for aperiodic streams have to be acceptable and conversely, the disk-access time of aperiodic requests should not infringe upon the guarantees of real-time streams.
- **Fairness Criteria:** Provisioning may be done to ensure that all these types of applications get a fair share of disk transfer bandwidth
- **Maintenance:** The requirement is to monitor the data rates being provided to real-time streams with respect to the guarantees provided beforehand, so as to perform QoS re-negotiation in cases of shortfall.

## 5. Survey of QoS End Systems

In this section, we shall survey the following QoS End System Architectures – Chorus [3], WashU [4], Omega [5], and HieRAT [6]. These systems are compared against our proposed framework and conclusions drawn based on it.

### 5.1. CPU Scheduling

#### 5.1.1. CPU Scheduling for Chorus

Chorus has a split-level scheduling mechanism - to minimize the effect of context switching overheads, scheduling is done at two levels: User level and Kernel Level. User Level scheduler schedules lightweight user level threads on top of a kernel level thread scheduled by the kernel level scheduler. It is ensured that the thread with globally the highest priority thread with the earliest deadline is scheduled. This strategy minimizes context-switching delays and meets real time criterion [3].

The total CPU utilization is partitioned between Guaranteed ($U_G$) and Best Effort ($U_B$) classes as given by:

$$U_G + U_B = 1$$

($U_G < U_B$ generally to encourage Best Effort traffic). This partitioning ensures that best effort class does not get starved out of CPU resources in the presence of guaranteed (higher priority) traffic – a mechanism to ensure fairness. Non real time threads in the system are given appropriate scheduling so that they have reasonable response times. This is also an instance of fairness.

However since utilization is reserved for guaranteed services and it may be so that there may not be guaranteed applications running always to full capacity while best effort applications are overbooked. Maybe $U_G$ and $U_B$ could be configured dynamically to maximize throughput.
Admission Criteria evaluate the requesting application based on application class and requirements and current CPU utilization for that class.

### 5.1.2. CPU Scheduling for WashU

The process management here has been addressed from the perspective of achieving maximum efficiency of communication protocols. Reduction in context switching time and minimizing data movement across protection domains are seen as the key components towards achieving that aim [4].
Protocol threads are implemented as part of user space along with application threads (unlike OS like UNIX where it is part of the kernel) and split-level scheduling is used to effect cost savings in terms of context switching. The periodic RT-thread model is chosen to describe CPU requirements and Rate Monotonic (RM) has been chosen as the scheduling policy that imparts real-time scheduling guarantees required by the multimedia threads in the system.
A technique known as *batching* is suggested for protocol threads, wherein batches of protocol data units (PDUs) are processed per invocation of the thread. Although this technique minimizes the context switches, there is a trade-off involved with thread period and thread quanta, that merits consideration in light of application requirements and preserving the QoS guarantees for other threads in the system.

Another idea presented for minimizing the context switches is to always ensure that a minimum number of PDU's are always generated whenever the thread is run. To this effect, delayed pre-emption has been added to the RM scheduling policy wherein a thread is allowed to run until it generate/consumes an integral number of PDU's even if a higher priority process may become ready while it is running. The generated PDU's are then enqueued at the network interface and the CPU is yielded in a single context switch. Further, in this implementation, the kernel scheduler does not preempts the running thread inspite of its losing its priority but only delivers a notification of the same. This is an *optimistic* approach that relies upon the running thread to relinquish the CPU voluntarily. This approach also forsakes any explicit policing requirement of CPU usage by the kernel scheduler. It may not be a good option to make the scheduler bereft of such a policing mechanism in order to ensure fairness.
Shared memory is used between the protocol threads and the network interface to store both the incoming and outgoing PDU's, to avoid *double-copy semantics.*
Admission criteria in taken care by the *schedulability test* that takes into account the modification (delayed pre-emption) introduced to the basic RM policy.
WashU does not address the issue of fairness criteria, owing to the lack of degree of service commitment considerations (guaranteed, best-effort etc.) in its multimedia application classification.

#### 5.1.3. CPU Scheduling for Omega

Omega is implemented as a middleware software running over AIX operating system and using the extension support, which consists of RT priorities with fixed-priority scheduling. It runs in user space and the extensions do not provide direct access to the AIX scheduler, so the scheduling is split as user scheduling and kernel scheduling (native AIX scheduler). The user scheduler implements a non-preemptive EDF algorithm. The advantage of non-preemptive algorithm is that there are no overheads for suspending messages for multimedia applications, in order to transmit a higher priority message. However, it could lead to *priority inversion*, when a higher priority message is blocked by a long lower priority message [5].

Omega also discusses a concept of *joint scheduling* to guarantee real-time requirements. A Global Precedence Graph is made from time-dependencies between events and scheduling is done accordingly.

Omega does not address the issue of fairness since it does not even define application classes. This is probably because it was implemented with very specific telerobotics applications in mind that did not call for such a classification.

As part of admission criteria, Omega takes the system QoS parameters and checks whether the task is schedulable. The QoS is agreed upon with the help of a *QoS Broker*. The QoS Broker can be invoked for any further re-negotiation of the QoS. However, Omega does not discuss any policies for policing or maintenance.

#### 5.1.4. CPU Scheduling for HeiRAT

The CPU scheduler in HeiRAT is based on pre-emptive, fixed priority RM scheduling. Applications are classified as Critical Guaranteed, Critical Statistical, Non-multimedia and Workahead with priorities in that order. The lower priority threads are left in wait-state until their logical arrival time, to avoid hampering higher priority threads, thus achieving real time criteria [6].

The admission criteria perform a throughput test for the CPU to determine whether the maximum throughput of the requesting application exceeds the remaining CPU bandwidth.

A watchdog policy exists to prevent violation of QOS guarantees in the presence of errant process. A certain degree of policing is achieved this way.

### *5.2. Memory Management*

#### 5.2.1. Memory Management for Chorus

In Chorus, the memory related parameters are

i) Number of buffers to allocate

ii) Access Latency

Buffers are allocated in accordance with the quantum of the thread on CPU, network delay and jitter limit. Memory is apparently not a bottleneck here, in fact it is used to compensate degradation of other resources e.g. jitter smoothening buffers. This brings out co-operation between resources towards meeting guarantees [3].

Pages are divided among guaranteed and best effort applications to ensure fairness. Once admitted, pages for guaranteed applications are locked in memory. In this case access latency is

bounded but for best effort applications, some may page fault and the latency depends on the page fault handler. There does not seem to be a specific upper bound on access latency.

There are admission test policies for admitting new applications are also based on their application class.

The *QoS Mapper* maintains virtual memory mapping tables and is implemented in kernel space to minimize context-switching overhead. The Mapper efficiently re-maps memory regions to minimize data copying semantics. It also maintains a statistics of page faults for best effort applications and attempts to keep the right pages in memory and this minimizes latency.

However, it is not clear how Memory Management handles buffer overloads since resources like I/O may also request for buffers which is not accounted for during allocation.

#### 5.2.2. Memory Management for WashU

Memory management has not been given adequate attention in WashU, except for a mention of the relationship that the application-specific QoS parameters (e.g frame size, frame rate) will directly translate into required maximum application buffer requirements.

#### 5.2.3. Memory Management for Omega

There is a buffer allocation test for both application and network tasks at the end-point supporting a QoS stream. It is basically to reserve buffers for the session during the call setup phase, and smooth the traffic jitter. However there is no mention of how pages are locked, and memory access latency is guaranteed.

#### 5.2.4. Memory Management for HieRAT

Memory management has not been considered substantially in HeiRAT, the only mention is that of the need to calculate the amount of storage needed and to perform a corresponding static buffer allocation at connection establishment time.

### *5.3. Network Interface*

#### 5.3.1. Network Interface for Chorus

The network interface parameters are bandwidth, delay bound and cell loss rate (since Chorus is ATM-based). Admission testing for network interface takes place in three steps to ensure real time guarantees:

- Bandwidth test: Checks for capacity at each switch to accommodate new application.
- Delay bounds: Checks of sum of local delays at each node is lesser than total network delay laid down by resource manager.
- Buffer availability checks for memory support at each switch for buffer allocation.

During run time, a cell-level scheduler tracks the frames sent out into the network and orders them on the basis of priority and deadlines. There is no such ordering for receipt of frames from the network since that implemented in hardware.

#### 5.3.2. Network Interface for WashU

Network interface performs the important task of scheduling the network link bandwidth for the transmission of

outgoing data. This scheduling implementation is facilitated for ATM-like networks wherein data is sent in form of fixed-size *cells*. The timing of cell transmissions is controlled using a pacing scheme referred to as HRR[4]. This pacing scheme enforces the peak bandwidth (derived from application QoS requirements) which ensures that applications never infringe upon their allocated bandwidth.

The admission criteria seems to be implicitly provided by the HRR scheme which would reject any new connection if the requested bandwidth cannot be obtained at any of the levels.

This pacing scheme also has the ability to control both the queuing delay and the cell-output rate on the output network link, which provides the real-time guarantees.

#### 5.3.3. Network Interface for Omega

Admission criteria for network interface in Omega are based on throughput test and End-to-End delay (EED) test. Throughput test checks whether the required bandwidth added to the currently allocated bandwidth remains within the upper bound bandwidth supported by the network. The network EED test checks for the duration of network tasks against the required end-to-end delay bound (in similar lines to CPU scheduling).

A Rate control test needs to be introduced in Omega to check for the number of network packets per second moved from user to kernel space and vice versa. This is because, there is an overhead of user buffer to kernel buffer copy when a network packet is sent. This inefficiency is probably because the MM extensions in Omega is implemented in user space, and drivers are implemented in native AIX OS in kernel space.

#### 5.3.4. Network Interface for HeiRAT

Since the order in which packets are sent to the network is determined by an internal scheduler (generally FIFO) and is not changeable, an additional external scheduler is built as an additional software layer which submits packets to the network adapter in accordance with their urgency. This is done by limiting the number of packets cached at the adapter. The external scheduler maintains the queue of packets to be transmitted. The packets from guaranteed streams have the highest priority followed by statistical streams with normal packets ranking last. With the first 2 classes, a real time EDF or RM is used, through the scheduling here is non-preemptive once a packet is submitted, its transmission cannot be aborted.

HeiRAT implements an external scheduler in a Token Ring network using the priority scheme of the MAC layer.

## 6. Summary and Discussion

We present a tabular summary of the provisions in the surveyed architectures with respect to the criteria in our proposed framework detailed in Section 4. Further, we discuss some future research directions that have stemmed from our survey. .

### 6.1. CPU Scheduling

| | Real time criteria | Throughput criteria | Admission criteria | Fairness criteria | Maintenance and Policing |
|---|---|---|---|---|---|
| Chorus | √ | √ | √ | √ | √ |
| WashU | √ | √ | √ | X | X |
| Omega | √ | √ | √ | X | X |
| HeiRAT | √ | √ | √ | X | √ |

### 6.2. Memory Management

| | Real time criteria | Admission criteria | Maintenance and Policing | Fairness criteria |
|---|---|---|---|---|
| Chorus | √ | √ | X | √ |
| WashU | X | X | X | X |
| Omega | X | √ | X | X |
| HeiRAT | X | √ | X | X |

### 6.3. Network Interface

| | Real time criteria | Admission criteria | Fairness criteria | Maintenance and Policing |
|---|---|---|---|---|
| Chorus | √ | √ | √ | √ |
| WashU | √ | √ | X | √ |
| Omega | √ | √ | X | X |
| HeiRAT | √ | √ | X | X |

### 6.4 I/O Systems

| | Real time Scheduling criteria | Resource Reservation | Admission criteria | Fairness criteria | Maintenance |
|---|---|---|---|---|---|
| Chorus | X | X | X | X | X |
| WashU | X | X | X | X | X |
| Omega | X | X | X | X | X |
| HeiRAT | X | X | X | X | X |

In our survey, we found inadequate treatment/attention provided to the I/O subsystem issues, which we think is an area of improvement for providing truly end-to-end QoS guarantees.

Memory management maintenance and policing mechanisms may prove to be useful in mitigating memory buffer overload scenarios by identifying the major offending processes, and enabling taking some action against them, thus preserving the QoS guarantees of the other processes in the system. A possible way to achieve this could be to *signal* the offending processes, which may serve as a directive to them to release all their memory buffers and terminate.

An interesting insight into subsystem dependencies is brought about in this

following example. Non-deterministic delay bounds in memory access due to page faults, etc. may lead to the thread getting swapped out, and the CPU scheduler needs to re-compute the priority of the thread to meet its deadline. This is clearly showing a *domino* effect where a performance hit in one subsystem (memory) may lead to further performance hit in another (CPU scheduling – overhead of re-computing priorities, rescheduling the thread again). This clearly demonstrates the need for various components of the system to work in harmony to achieve the end guarantees. Further research may unravel critical subsystem interactions that need to be addressed for superior system performance.

Memory caching for multimedia streams are driven by different considerations than the traditional *locality of reference* or working-set characteristics. Existing page replacement policies like LRU, LFU, MFU, etc. are unlikely to be as effective for multimedia streams as they are for other types of applications. This suggests a future research area to explore effective page replacement and caching strategies well suited for multimedia applications.

## 7. Conclusion

Supporting Quality of Service (QoS) requirements for multimedia applications places significant demands on Operating Systems as a resource manager for managing end-system resources. In this paper, we approached the key issues involved in various subsystems (CPU scheduler, Memory, Network Interface, I/O) towards meeting these QoS requirements. We did so by characterizing a framework for a QoS-Aware OS that includes policies for static and dynamic resource management. We also studied and analyzed some existing implementations (Chorus, WashU, Omega, HeiRAT) and evaluated them against our framework.

## 8. References


*1.* Ralf Steinmetz. Analyzing the Multimedia Operating System. *IEEE Multimedia,* Spring 1995
2. Campbell,A., Coulson,G. and D.Hutchinson. A Quality of Service Architecture. *ACM Computer Communications Review*, April 1994
3. Coulson,G., Campbell,A., Robin,P., Blair,G., Papathomas,M. and D.Hutchinson. The Design of QoS controlled ATM Based Communications System in Chorus. *IEEE JSAC Special Issue on ATM Local Area Networks*, 1994
4. Gopalakrishna,G., and G.Parulkar. Efficient Quality of Service in Multimedia Computer Operating Systems. *Department of Computer Science, Washington University Report WUCS-TM-94-04*, Aug 1994
5. Nahrstedt,K. and J.Smith. Design, Implementation and Experiences of the OMEGA End-Pont Architecture . *Technical Report (MS-CIS-95-22), Univ. of Pennsylvania*, May 1995
6. Volg,C., Wolf,L., Herrtwich,R. and H.Witting. HeiRAT - Quality of Service Management for Distributed Multimedia Systems. *Multimedia Systems Journal*, 1996
7. A.L.Narasimha Reddy and J.Wyllie. I/O Issues in a Multimedia System. *COMPUTER,* 27(3) :69-74,1994
8. Clifford Mercer, Stefan Savage and Hideyuki Tokuda. Processor Capacity Reserves: Operating System Support for Multimedia Operating Systems. *Proceedings of the First IEEE International Conference on Multimedia*

*Computing and Systems*, pp. 90-99,Boston,MA,May 1994

9. D.James Gemmel, Harrick.M.Vin, Dilip.D.Kandlur, P.Venkat Rangan. Multimedia Storage Servers : A Tutorial and Survey. *IEEE Computer*, 1995

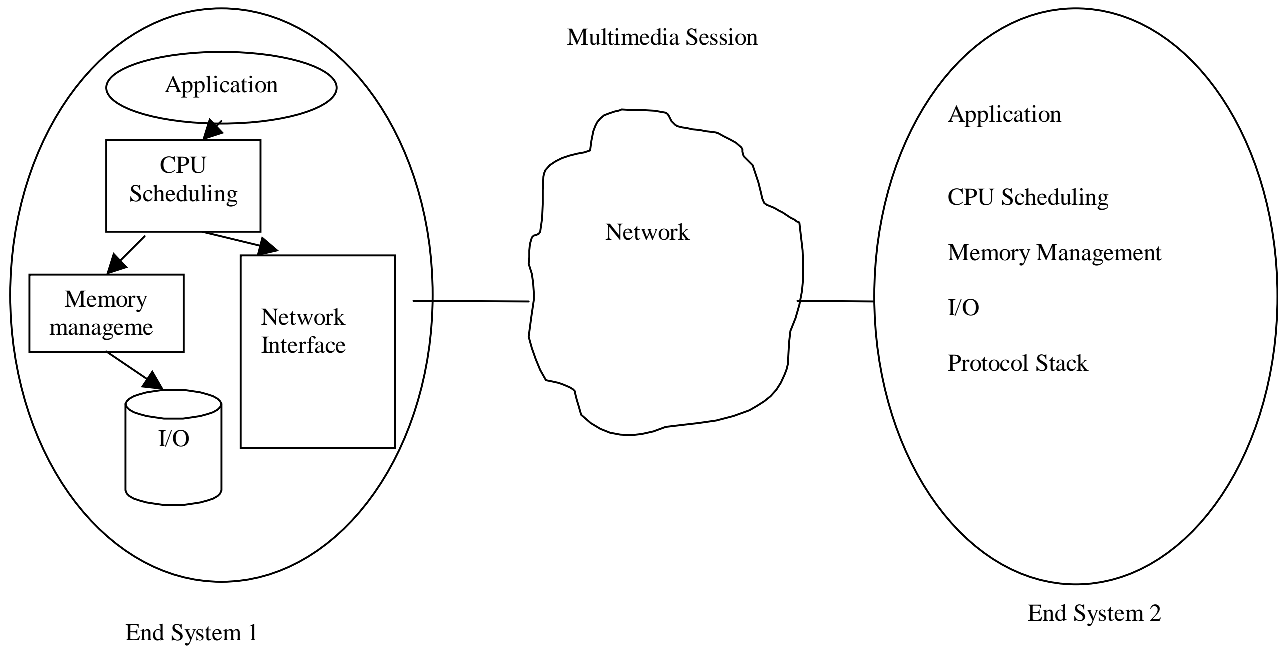


**Fig 1 : QoS End-to-End System**

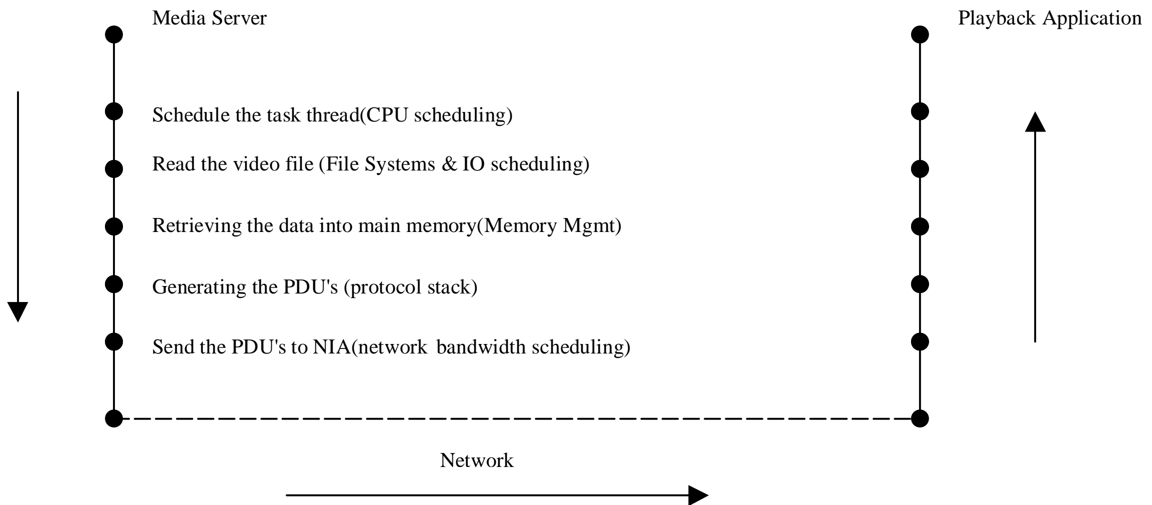


**Fig 2 : QoS End-to-End Activity Diagram**

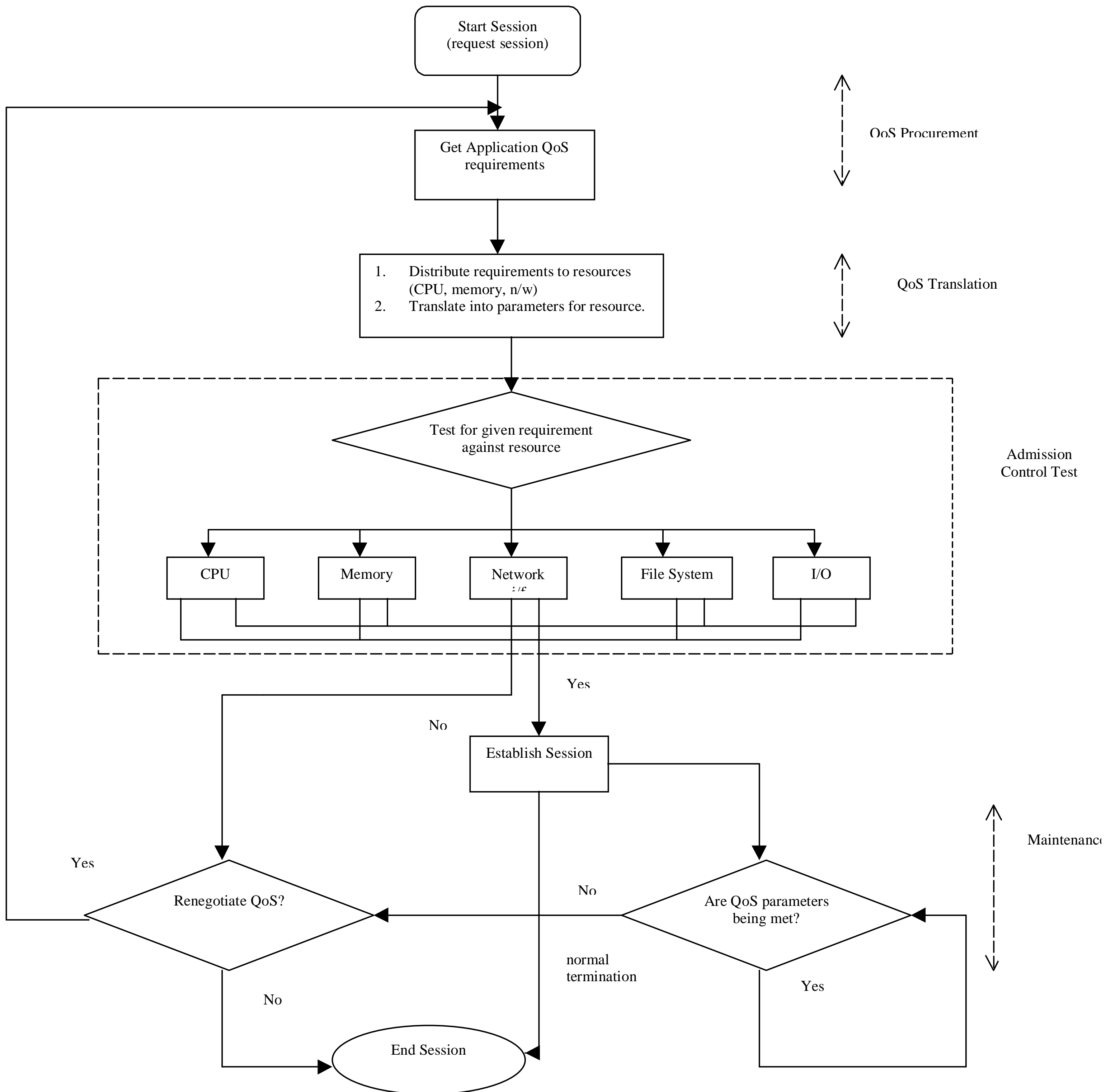


**Fig 3 : Flowchart for Generalized QoS Mechanism**