CHAPTER-VI

OBJECT ORIENTED ENGINEERING AND DISTRIBUTED QoS APPROACH FOR MEDIA STREAM (Me-ADO-Q)

6.1 Introduction

The advent of next-generation technologies such as mobile with complex media delivery over broadband Content Delivery Networks (CDN), are expected to be highly scalable[3,8,9,59] and adaptive over a wide variety of heterogeneous devices and distributed environments [37]. Such objectives stress on need for significant extensions to current generation, distributed multimedia technologies, to provide an end-to-end Quality of Service for streaming media. To provide guaranteed end-to-end service quality[26,10], the streaming system requires resources, such as storage, network bandwidth and buffer allocation to manage the streaming media for each media session.

Me-ADO-Q (Object Oriented Engineering and Distributed QoS approach for Media Stream) focuses on providing Quality of Service by controlling and inter-coordinating among system resources, to satisfy the heterogeneous nature of various distributed multimedia applications (audio, video, images from Shockwave, Flash). The resources are mapped together to provide coordination, such that multiple resource components of operating system, communication system and other middle-tier functions can work on providing optimal QoS.

Me-ADO-Q, an object based distributed engineering environment is developed using CORBA\(^1\) under Linux environment using Java's Interface Description Language [56]. The simulated model of Me-ADO-Q is developed

\(^1\) UML Reference – Refer Appendix A
in Linux and tested. This work is primarily limited to monitor and control media streaming data. Me-ADO-Q model can also be represented in short as M-Q.

6.2 Need for End-to-End QoS in Media Streaming Networks

The stochastic behavior [4] of multi(ple) media streaming network requires guarantee of QoS for time-critical applications (such as on line Video Telephony[58], Video Conferencing[65], Video On Demand applications[56]). Streaming media consist of single, periodically changing continuous bits, (such as audio samples or video frames) which should provide effective quality of transfer as expected by user. The concept of user expectancy is being defined as "acceptance of quality" or Optimal Service Quality (OSQ). To achieve, the system must satisfy various heterogeneous requirements along an end-to-end path under different real-time conditions and communication constraints of run-time computing environment. To guarantee on the parameters of processing, communication and delivery of desired media quality to the user, MQ model should work on:

1. Providing scalable and adaptable media quality on network.
2. Resources required for process during run-time should be available to minimize the delay and manage "quality".
3. Should strive on providing quality even in worst-case situations.
4. Minimize end-to-end delay in providing quality.

DiffServ [49] and IntServ had discussed on delivering QoS for networks but they do not focus on "providing quality " for media streaming applications. DiffServ has been accepted as a better alternative of IntServ for managing
traffic, but do not provide an end-to-end QoS. It provides reduced throughput as it does not follow any better policy managed admission control schemes. IntServ concentrates mostly on signaling between devices and few network QoS parameters. RSVP is a good approach of reserving required bandwidth for providing quality, but if demanded bandwidth or resources are not available, then desired quality cannot be provided. Both IntServ and DiffServ does not focus on providing end-to-end QoS support.

General properties required for end-to-end QoS are:

1. Predictably fast response [68] and media synchronization for mission time-critical events with accurate timing and information.
2. High degree of schedulability [80], and degree of resource utilization for which the deadline of each time-critical task should be accountable.
3. Stability [86] under transient overload and system overload, where the processing of critical tasks must be ensured.
4. Multi-tasking [96] is required. A real-time application with various individual tasks, should co-ordinate simultaneously to reduce delay.
5. Fast context switching [94] represents the time interval when OS recognizes a new process or thread (a waiting process / thread in the process queue) needs to run, with all required components.
6. Rich set of Inter-task communication mechanisms [105] is a required feature. Time-sensitive IPC mechanisms such as message queues, shared memory, semaphores and event flags are required.

Basic parameters required for providing QoS are in Table 6.1.
Providing an end-to-end QoS is a challenging goal in multimedia networks. The complexities in providing QoS are as follows:

i. User expectations [116] of multimedia processing and communication in a digital distributed environment are always compared with the TV quality and radio quality networks for clarity.

ii. Should provide heterogeneity of OS services [115] (e.g., Windows, UNIX, Linux, Windows CE) running on different platforms (e.g., SUN workstations, Pentium PCs, handheld devices).

iii. Heterogeneity of network and transport protocols [115,121,122] (e.g., RSVP, IPv6, RTP, RTSP, HTTP, UDP, TCP) running on different networks (e.g., Gigabit Ethernet, 100 Mbps Ethernet, ATM, 802.11 wireless networks, Blue tooth).

iv. Different administrative domains and federated systems with varying policies and security restrictions should be employed.
To achieve this goal, above presented QoS features and resource management must become an integral part of distributed multimedia system.

6.2.1 Provisioning End-to-End QoS

Traditional QoS [118,123] is defined for the network layer of the communication system. In streaming multimedia systems the QoS concept should be extended because of multiple services beyond network and transport services contribute to delivery of end-to-end quality. The set of chosen parameters for the particular service determine the measure of QoS. ISO [95,106] has defined a set of reliable parameters, which measure network transfer quality such as throughput, delay, error rate and cost. Some other common QoS insistent “quality” parameters are connection establishment delay, connection establishment failure, throughput, transit delay, and residual error rate. Different levels of QoS required for providing an end-to-end QoS are:

1. Perceptual Level (Higher level QoS such as Media Quality)
2. Application Level
3. System Level (The middleware components and OS Level)
4. Network Level
5. Device Level

The Perceptual Layer QoS or User QoS is the quality expected from User's end. This QoS measure focuses on response time of media data, quality at media receiving end based on media device and pricing issues.

Fig-6.1 shows the various layers required for providing end-to-end QoS.
Fig-6.1 QoS layered model for multimedia networks

**Application Layer QoS** parameters describe requirements for application services such as media characteristics (e.g., frame rate, frame resolution), their transmission characteristics such as delay, jitter, media relations, and specifying relations among media transformation with inter and intra frame synchronization.

**System Layer QoS** parameters focus on requirements of communication and computing services, derived from application QoS. Qualitative criteria specify the expected functions needed for provision of QoS such as inter-stream synchronization, ordered data delivery, error-recovery mechanisms, scheduling mechanisms and others.

**Network Layer QoS** parameters describe requirements placed on low-level network services. Network load describes the ongoing network traffic identified through average / minimal inter-arrival time on the network connection, "burstiness", packet size and service time. Performance might be expressed through delay-time for media packet play-out or packet loss rate. Inter-arrival of media packets also carries probable importance on QoS parameters.
Device Layer QoS parameters typically specify timing and throughput demands for media data units given by audio/video devices such as audio speakers or video capture boards.

Device QoS and User QoS are inter-related, but generally output quality depends on the input quality of system. If input data quality is poor, then, even if the service has plenty of resources available for processing and communication, the output quality of the service will be poor. To fulfill the timing requirements of continuous media, the operating system must use real-time scheduling techniques. These techniques must be applied to all system resources involved in continuous media data processing, i.e., entire end-to-end path is involved.

To achieve an acceptable level of QoS measure, an Optimal Service Quality (OSQ) is defined as $\alpha$ in Me-ADO-Q model. Systems must negotiate and identify resources under heterogeneous communication environment with varying conditions and constraints of run-time computing, with optimal QoS service value to deliver required quality of media stream.

6.2.2 Modeling an End-to-End QoS

Variable service parameters are being called to provide an end-to-end QoS. Effective quality of service defines call receiver’s $MQ_r$ value accepts the call originator’s $MQ_c$ value, with set of parameters to satisfy the required OSQ parameter in providing quality. The parameter is acceptable under the condition that media data would be transferred within the specified time interval ‘t’ without any change in policy parameters. In service architecture, based on condition of ‘$\lambda$’ and ‘$r$’, different approaches are adopted:
1. Best Level QoS
2. Deterministic Level QoS
3. Acceptable Level QoS
4. Peeved Level QoS

Best Service provides the required service as requested by User. Acceptable, and Deterministic Services enable optimal service of media packet delivery based on required OSQ, while Peeved Service provides delivery but cannot guarantee the requested quality of OSQ. Me-ADO-Q Catalyst handler helps in providing quality under worst situations. Delivering the expected quality is based on following parameters:

- \( t \) is expected time interval (in ms) of media data at receiver’s end.
- \( r \) is a real positive integer representing the effective resources available.
- \( \lambda \) variable parameter required for acceptable error limit in providing quality, vector in quantity.
- \( \alpha \) optimal service point - specifies the required quality, vector in quantity.

Here \( \lambda \) is a small quantity whose intensity ranges in order of \( 10^1 \) to \( 10^3 \). The constraints are provided as \( MQ(\lambda, r, t) \) as a set of variable parameters that help to determine the type of QoS.

a. Best Level QoS

QoS guarantee can be achieved between the acceptable QoS level and available QoS value that can be provided. \( MQ_c(\lambda, r, t) = \alpha \) defined for guaranteed, best level media transfer. Here \( \lambda \) being a set of variable vector end-to-end QoS parameters. \( MQ_c(r) = MQ_z(r) \), hence \( MQ_c(\alpha) \leq MQ_z(\alpha) \). \( \lambda \) is of order of \( 10^{-3} \) to \( 10^{-2} \). Here all the required resources \( r \) are available to provide required quality at specified time \( t \).
b. Acceptable QoS

For any call session, $MQ(\lambda, r, t) > \alpha$ then utilization of resource and parameters are identified but the expected quality $\alpha$ cannot be guaranteed, here $MQ_c(r) > MQ_r(r)$. An optimal quality service '$\lambda$' can be generated which can be accepted by both call originator and call receiver, which is based on 'r'. '$\lambda$' is of order of greater than $10^{-2}$ and less than $10^{-1}$.

c. Deterministic QoS

The overall deterministic limits of lower bound and higher bound positive values are to be defined for guaranteed media transfer. An ubiquitous network can be working above the higher bound at any acceptable policy, while guarantee policy level for media transfer when required QoS parameters are "Not In Reserve" to be assigned at an instant. If any $MQ(\lambda, r, t) \sim \alpha$ then utilization of available resource 'r' and its relative parameters are not properly identified, but MQ tries to transfer the information within specified time, since $MQ(\lambda, r, t)$ is well within $\alpha$ OSQ value. Here $MQ_c(t) > MQ_r(t)$ and $MQ_c(r) > MQ_r(r)$ for any value 't'. $\lambda$ is of order $10^{-1}$ to 0.

d. Peeved QoS

$\lambda$, $r$ and $t$ does not fall into MQ category and OSQ value, it is far below the limit. In peeved QoS model quality cannot be guaranteed with any resource allocation. Here $MQ(\lambda, r, t) \neq \alpha$ where $MQ_c(t) \neq MQ_r(t)$, $MQ_c(r) \neq MQ_r(r)$ while $MQ_c(t) - MQ_r(t) > 10^3(ms)$. $\lambda$ is of order 0 and higher.

6.2.3 Measuring QoS and QoS Metrics

To provide an end-to-end QoS for media streaming network various measures to identify quality are discussed. A policy manager carries out the
transfer of media streaming operation over networks on an agreeable QoS measure. The guarantee of QoS on acceptable QoS is set of policies being agreed between the call originator and call receiver. In Me-ADO-Q, OSQ is an agreeable parameter that bridges the service components and resource parameters of the call originator setup and call receiver. An autonomous service consists of a set of functions, which accepts input data with QoS based OSQ value \( MQ_{in} \) and generates output data with OSQ value \( MQ_{out} \), both are vector types. In order for the autonomous service to process quality based OSQ \( MQ_{in} \) and generate output data of quality \( MQ_{out} \), specific resource \( 'r' \) is required. \( 'r' \) is a real positive value of different resources \( r = [r_1, r_2... r_m] \) (e.g., Processor speed, memory, bandwidth) which are required for QoS setup with following constraints:

a) \( 't' \) is the expected time to complete the process of receiving the media data. The received media stream should neither be earlier than the responding period and not delayed than the expected period.

b) \( 'r' \) denotes the set of available resources for providing the quality which includes device parameters such as frame resolution, video or audio noise less device.

b) \( MQ_\alpha \) denote the media quality being received, where \( \alpha \) is a vector value defined as positive value to denote the quality.

The test-bed setup is prepared using ITU Recommendation E-Model approach. Bursts of consecutive lost frames, jitter, and one-way delay measured in the test is used for calculation of a MOS estimate for E-Model. The E-Model was extended to identify the percentage of packet loss, packet
loss burstiness (calculated from maximum consecutive packet loss), the jitter buffer, and the stream characteristics.

6.3 Me-ADO-Q

Me-ADO-Q model (Fig 6.2) represents intra-communication among various QoS resources to provide quality. Me-ADO-Q defines a set of multiple objects that have variable properties and corresponding events, which monitors, controls the variable parameters striving to provide QoS. Each object intra-communication adopts an end-to-end distributed management model. Due to the heterogeneity of requirements, the services of various multimedia components must be object parameterized. The user/application requirements on multimedia systems are viewed as objects, which are mapped into various services and events of objects to provide quality and satisfy the requirements.

<table>
<thead>
<tr>
<th>Policy Generation</th>
<th>Activation and Management Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource Discovery and Service Discovery through Active Agents</td>
<td>Run Time adaptation Services through Active Agents</td>
</tr>
<tr>
<td>Service Level Agreements</td>
<td>Service Management and Disaster Recovery</td>
</tr>
<tr>
<td>Operating System and Network Protocol Stack</td>
<td></td>
</tr>
</tbody>
</table>

Fig-6.2- Me-ADO-Q General Architecture

Me-ADO-Q is an object based engineered approach where each component is described as an object. Object is represented by its id MQ-ID. This model identifies QoS parameters that require priority at run time adaptation and maintenance by using Agent discriminate functions. The
model also predicts the quality at runtime that should be delivered to end user with QoS metrics.

Policy Manager carries out a set of rules to decide on quality of media with QoS parameters. At each stage, the framework enables the interaction of peer middleware components called as agents across host boundaries, so that the provisioning of an end-to-end QoS can be performed in a coordinated and coherent fashion. Me-ADO-Q is discussed as framework of a distributed agent based architecture, with a case study.

QoS middleware components striving to achieve scalability and adaptability in media streaming networks is discussed by Me-ADO-Q. By interacting with operating system kernels, application level architecture, adaptive QoS can be provided for terminal end support. The Adaptive nature insists on learning of application hooks, iterative procedures and memory allocation libraries, which help to learn the application behavior at runtime. Knowledge of application configuration, memory address, threading, learned through MQ agents (Fig-6.3,Fig-6.4) about the resource availability, knowledge of resources helps in providing guaranteed QoS expected by user.

6.3.1 Me-ADO-Q Stack

The stack architectural model of Me-ADO-Q for providing end-to-end QoS is shown in Fig-6.2. Services are performed on objects, such as media sources, network connections and sockets\(^1\), with Me-ADO-Q QoS functional handler. The complete set of required QoS objects to provide QoS guarantee for an end-to-end provisioning is termed as M-Q Entity. An entity consists of

\(^1\) Table 6.1
all parameters from Operating System level parameters to User Level and Device Level QoS parameters defined within a M-Q Domain (Fig-6.3). The stack architecture shows an ordered level (top-to-bottom or bottom-to-top) of communication among modules. Network Protocols, Network Architecture, Operating System and basic platforms are bottom layer or first layer of M-Q. Middle-Tier elements and other third party modules where various service agreements and resources are identified from the second layer. The third layer consists of policy generation schemes, and Adaptive modules. The fourth and last layer has installed Agent modules, which inter co-ordinate with other agent modules and provide run-time adaptation.

6.3.2 Me-ADO-Q Agent Components

The architectural model of Me-ADO-Q defines multiple software agents, which manage their neighboring agents within host boundaries. The primary work of M-Q agents is to monitor the activity of the host media network and communicate with other agents (Fig-6.3). Agents are the backbone of M-Q architecture; any number of agents can be created and installed dynamically depending upon the variable parametric objects of M-Q Entity. Each agent object has its specific set of properties. The agent communicates with other agents by issuing an intra-co-operative policy request approach. They are software modules, which are created when a resource component is identified during transit of transmission from source device to receiver device. These agent objects follow the property of inheritance, which adheres to similar set of functionalities. Three agent objects are used in Me-ADO-Q work.
a. **Parametric Object Agent** – these agent objects are created dynamically depending upon the middleware components for any call setup process. If number of gateways increase for a call setup process, then gateway agent objects will be created by the system automatically with their corresponding set of parameters as properties. These agents work on object M-Q with the set of parameters.

![Mediator Agents Interaction Diagram](image)

**Fig – 6.3 Me-ADO-Q Object Agent Interactions**

b. **Mediator Agents** (Fig-6.3) are assigned as intermediate agents hemmed between groups of Parametric Object Agents serve as a manager controlling the signaling activities of other agents and obtaining the parametric values from other agents for discussion.

c. **Me-ADO-Q Catalyst Agent** is an active agent, which provides an alert to each neighboring agents within M-Q entity. This is an alternative solution, which provides solutions to handle complex media transfer which may lead to failure. Request and reply process uses simple probing method used for signaling between each agent, who pass the information to Mediators Agents. Mediator Agent collects the required parameters at each agent and analyses of resource requirements are carried out. The result is submitted to Policy Generator Module which in turn calls appropriate Policy Negotiator procedure for policy to be negotiated and adapted in run time process. The policy generated is...
provided to call originator and call receiver device for providing the scheduled end-to-end QoS.

Fig 6.3 presents an agent level intra-communication process where \( a_1, a_2, a_3, a_4, a_5 \) are various agents of Call originator end device. These agents gather the corresponding QoS data by service and generate policy from the required QoS Entity. Similarly \( b_1, b_2, b_3, b_4, b_5 \) are agents of call receiver, which intra-communicate with other agents and generate policy for the quality required.

### 6.3.3 Service Component - QoS Provision Architecture

Servicing procedures focus on processing and communication methods to provide quality guarantee during the complete session taken by streaming media to process and communicate within applications. Servicing focuses on time-sensitive requirements and high-throughput for applications, being sustained for long period of time. It also focuses on difficulties being caused by different classes of the network traffic produced by heterogeneous multimedia applications during transfer.

![Diagram](image.png)

**Fig 6.4-Servicing procedures on Objects**

Servicing is performed on objects, such as media sources, connections and virtual circuits (VC), which are object parameterized as \( C_1, C_2, \) and \( C_3 \). Hence, QoS parameterization reflects on the service objects that act on
objects (Fig-6.4). Events are functions generated by component objects on which the services are called into access handled by Me-ADO-Q Policy Generator. In operating systems, the service objects are number of tasks [26], memory chunks [26] which are characterized with quality parameters such as deadline, processing time, memory size, access pattern, CPU bandwidth allocation and memory bandwidth availability. Similarly parameters such as semaphore, buffer allocation also carries importance. If buffer size at receiving end is limited then the media stream at call receiver end to “play out” increases abruptly thereby reducing latency effect.

6.3.3.1 Middleware Service Tracking Architecture

In middleware systems, service objects such as media distributed resources, remote invocation methods (RMI), device specific parameters and software components are labeled with MQ ID. It controls the object of distributed services in timely fashion during their configuration, distribution, mapping and processing phases. In ISO–OSI Network architecture, network layer and transport layer require QoS identity. The transport layer QoS parameters include connection establishment delay, connection establishment failure, throughput, transit delay, residual error rate, transfer failure probability, connection release delay, connection release failure probability, protection, priority, and resilience.

Any middleware component modules, such as heterogeneous client platforms different networks architectures, varying bandwidth, processor speed, conversion trans-coders and other multi-formats of media are adopted. Similarly for audio its corresponding formats are embedded with several tracking filters (Section-6.7) middleware components include database setup,
network protocols, and external parameters such as echo suppression, noise reduction, which play vital role in providing quality.

6.3.3.2 Resource Discovery

Each object and agent of Me-ADO-Q provide continuous "Querying" or "Learning" over its resources to identify new resources, change in resource, status, settings, device configuration, change in user location during runtime. The framework considers real time media streaming to be invoked at runtime instantiation, disregard of any resource reservation procedures. The system uses SNMP [12] probe based signaling to identify the resources at runtime. This protocol monitors and identifies change in over all system at any layer of Me-ADO-Q stack. Resource Discovery also includes maintaining Resource Mapping Graph called MQ-Graph (Fig-6.5).

Me-ADO-Q model classifies resources into two types as Active Resources and Passive Resources. Active Resources are sensitive and adaptive to real time media streaming environment. Active resources are queue buffer allocation, change of user location, type of device in use, traffic changes, packet loss, and server activation intervals. Passive resources are not adaptive to run time environment, some resources are device status, type of media device in use, OS kernel, middleware components, and configuration changes. MQ model does not reserve any resource for the session as implemented in IntServ or DiffServ. The model assigns or shares a resource dynamically as required based on resource availability.

6.3.3.3 Me-ADO-Q QoS Parameters

Me-ADO-Q classifies QoS parameters as Imperative QoS parameter and Trivial QoS parameter. Imperative QoS parameters are sensitive during
adaptation runtime process, which alters the process in producing optimal results, hence given higher priority. Trivial QoS parameters do not provide any unexceptional changes in quality process. The parameters do not carry importance in providing QoS for provisioning architecture. Me-ADO-Q focuses on Imperative QoS parameters of different components such as memory handling of an operating system, threading features, semaphore handling, which decide on quality of media application.

Common media streaming network parameters are:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bandwidth</td>
<td>Identify the available bandwidth in network</td>
</tr>
<tr>
<td>Buffer</td>
<td>Capacity of path to allocate packets in queue</td>
</tr>
<tr>
<td>Path Class</td>
<td>Identifying media paths for Media Transmission</td>
</tr>
<tr>
<td>Packet Notifier</td>
<td>Type of packet (media / data)</td>
</tr>
<tr>
<td>Traffic Class</td>
<td>Media Priority Traffic Class</td>
</tr>
<tr>
<td>Packet Stamp</td>
<td>Time Stamp generated by sender</td>
</tr>
<tr>
<td>Virtual Path</td>
<td>Dynamic Buffer allocated on Demand from neighboring server.</td>
</tr>
</tbody>
</table>

6.4 MQ – Graph

A complete directed graph, MQ-Graph (Fig 6.5) is obtained for inter-coordinating between agents and other agents of system or application. Graph is maintained as a matrix in both call originator agent and call receiver agent with corresponding object properties and events in use. It also maintains the periodically selected agents for querying as well objects called into activation for specific period of time. Me-ADO-Q algorithm maintains the state of each object at run time. Similarly any change detected in object state on any hosts (call originator) is propagated to all active participation objects.
This graphical adoption provides a simple agent handling procedure by which the state of object at run time is maintained. MQ-Graph is maintained at each agent node. Graph status is renewed and redrawn for the whole session until call process is completed.

6.4.1 QoS Run Time Process

Me-ADO-Q follows various run time process to provide effective report on application configuration changes. Run Time procedures to be adopted are Querying, Initiation, and Adaptation. Querying at runtime provides an effective measure to monitor and configure complex multimedia applications. Learning is useful in gathering knowledge about application behavior, so that better run-time adaptation rules may be set accordingly. The process of querying is the responsibility of agent objects to probe into object's identity, identify their parameters and change in value. Change in status of a communication device or sudden changes of traffic intensity of WAN may contribute to changes in routing path of network or sharing of resource objects among other media applications or frequent user changes.

A multimedia application may have various application configurations. Each application configuration consists of a set of application components, which is represented by application configuration graph MQ-Graph residing in
each agent nodes. At the stage of run-time instantiation, the middleware will select a suitable configuration, which matches the specific resource availability and user preference. The agent middleware assists the application to adapt to the new or existing configuration including changes of resource availability or user requirements. Run Time adaptation provides application adapting to changes in environmental setup.

Fig-6.6 MQ- Resource Negotiation

Fig 6.6 provides a sample policy generation from Parametric Agent modules using three agent parameters buffer (B), bandwidth (P) and time limit (T) in Call Originators device. The agents adopt common responsive methods to learn from system. Based on the values obtained from each agent, the systems negotiate and decide on policy to be generated. The policy is sent to intermediate middleware agents such as Gatekeeper agent, Gateway agent or Router agents, which learn from other neighboring agents required OSQ value for providing QoS.

6.5 Policy Handler

Policy Handler is a rule-based generator module, which issues request, negotiates, alters and assigns the required parameters to provide quality by controlling the activity of objects and agents. Multiple objects inter-coordinate together on decision with agents to assign a resource or alter or refuse a
resource from activity. All decisive steps carried out by MQ is ascertained by Policy Handler. Various policy methods are being adopted to invoke agents and corresponding services.

a. Policy OSQ

The optimal service quality (OSQ) has to be initially determined by Me-ADO-Q on identifying the OSQ limit. Limit depends on call originators resources and resources required for call receiver, which can provide the required quality. The command to determine the OSQ is as follows:

\[
\text{OSQ\_Value} = \text{Policy\_OSQ} (\text{< Request\_MQ\_Parameters >}, \text{< Resource\_Value >})
\]

Here,

- **OSQ Value** - Return value for requested resources
- **Request MQ Parameters** - Callers request parameters
- **Resource Value** - Resource parameter values

OSQ value is a vector value represented in decimal format, based on resources in use, or resource value required.

b. Policy Negotiator

Policy Negotiation process entrusts identifying the required component parameters through agent information gathering, and concluding on the parameters for establishment of call. During application run time, the media stream has to undergo variations in accepting a resource or altering the resource.

\[
\text{Policy\_Negotiate} [\text{< Parametric Type = Parametric Value >}, \text{< Priority = Value >}]
\]

Here,

- **Parametric Type** - The types and values of QoS parameters used for negotiation
- **Priority** - Denotes the value of QoS variables

Policy Negotiation selects the parameters that are reserved for use both by call originator and call receiver. Negotiation process is carried out on common
set of required parameters that can guarantee QoS. Policy Negotiator scheme insists on providing QoS with available parametric values based on type of media application. Some Policy Negotiation parameters generally used are Bandwidth on Demand, Alternate Path for routing, Reserve Bandwidth, Packet Size

c. **Policy Request:**

Generates policy-request with set of QoS parameters for requesting the required policy to provide quality. The requested policy being generated should be accepted both by call originator and call receiver to carry out a call session between two or more terminal endpoints. The components required for setting up the call are reserved in respect of agent domains of call originator and call receiver, which ensures that the components selected are reserved for complete call session. Policy request command is shown below:

```
Policy_Request [ <Call_OriginatorID>, <Call_ReceiverID>,
                <Request_MQ_Parameters>, <OSQ_Value> ]
```

Here,
- **Call Originator ID** - IP Address of call originator
- **Call Receiver ID** - IP address of call receiver
- **Request MQ Parameters** - Parameters required for quality negotiation
- **OSQ Value** - Optimal quality value generated

If a call of higher hierarchy requests the components for call establishment, then component can be freed or allotted for use on share basis. Corresponding update is done on MQ-Graph.

d. **Policy Adaptation:**

Adapts to any change in end-to-end environment. Change in resource parameters should be accepted both by Call Originator and Call Receiver. Instances such as change of resource status / availability, resource
reservation mechanisms, user preferences are primary constraints leading to policy adaptability at run-time. Adaptation policy also works when requested resource parameters are not available hence other alternate parameters are in request.

| Policy_Adapt [ < Resource_Available = TRUE | FALSE >, < Resource_Status = BUSY | IDLE | FAULTY | NO_RESPONSE > ] |
|---|
| Here, |
| Resource_Available - Availability of required resource, returns Boolean value |
| Resource_Status - Status of resource requested |

e. Policy Establishment:
Assigns the policy generated between two domains or gateways with acknowledgement from both ends (call originator and call receiver) for any call to be established between the end devices. Policy Establish command is shown below:

<table>
<thead>
<tr>
<th>Policy_EstabIish [ &lt; Request_MQ_Parameters &gt;, &lt; PortID &gt;, &lt; SocketID &gt;, &lt; CalleeID &gt;]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Here,</td>
</tr>
<tr>
<td>Request_MQ_Parameters - Parameters required for providing quality</td>
</tr>
<tr>
<td>PortID - Port number of Call Originator or Call Receiver</td>
</tr>
<tr>
<td>SocketID - Socket number for establishing connection</td>
</tr>
<tr>
<td>CalleeID - IP Address of call receiver</td>
</tr>
</tbody>
</table>

6.6 MeADO-Q Catalyst - Self Healing Approach

Perceptual studies have shown that the tolerance level for discontinuity in continuous media server [12] is beyond user's satisfaction level. Me-ADO-Q Catalyst scheme uses an end-user driven approach to minimize the amount of discontinuity perceived. Typical error handling schemes provide moderate solutions to handle fixed faults, also variable faults that are unexpected at run-
time such as handling congestion faults, session management which lead to link failure are considered.

When quality oriented MQ($\lambda$) goes below the lower bound limit of $10^{-3}$, policy negotiation is being carried out to transfer media data between the end devices on parameter agreeable policy. If negotiation being carried out does not provide an agreeable situation, then condition is considered to be worst situation where MQ QoS falls below the lower bound. Under this condition Me-ADO-Q Catalyst is called into action to provide persuasive solution for media streaming transfer. Catalyst handles faulty situations as follows:

a. Providing client end reservation and service schemes
b. Continuous monitoring of resources and events.
c. Managing resources in active state
d. Alerting the user in times of unavoidable fault, hence user selects alternative setup for continuing the call session.

If an active resource is reserved for use, then the resource is not being assigned only for the current ongoing session. Other parallel sessions which demand similar resource can also utilize the network resource if free at any instant. Guarantee of QoS for resources is considered and initiated before a session is established.

6.7 Me-ADO-Q Working Model

The algorithm Me-ADO-Q works on media packets and its parameters with corresponding set of values, which are obtained from various agent objects provided to mediator agents. Deciding on the path, buffer and required parameters for media transmission are carried out by Policy Negotiation. The negotiation and selection of required parameters may vary for each setup and
application, depending upon the environment available at instance. If the media transfer path possesses high traffic intensity, then an alternative path for transfer with required bandwidth should be selected according to Policy Negotiator Scheme.

Steps followed in providing QoS for Me-ADO-Q

1. Resource Discovery
2. Learning Resources and its characteristics by querying
3. Resource Notification
4. Policy Negotiation
5. QoS Parameters allocation and Resource allocation
6. Policy Generation (between egress end points)
7. Session Establishment
8. Media Stream Transfer
9. Monitor Resources

Fig 6.7 Me-ADO-Q Working Model
10. Session Orientation

11. On Alter Request - Session Reconnect / Policy Alter

12. Policy Regenerate

13. Session Ends

14. Policy Close (Free Resources)

---

Fig 6.8 Me-ADO-Q Policy Handler

Parameters used in Me-ADO-Q Working Scheme are:

1. No of Packets → Pk
2. Total required QoS media → TMq (parameters from Call Receiver)
3. Identify available Parameters Call Originator & Call Receiver.
4. Create Dynamic Object Agents with Service Agents
5. Create_Agents ( )
6. Call FeedBack_Notifier ( ) and Select Route_Path ( )
7. Call Traffic_Policy_Generator ( )
Redo Step 5

8. Generate PolicyAuthentication ( ) (router / gateway between two variable gateways and network domains)

Redo Step 5

9. Generate packets

10. Transfer packets

**Me-ADO-Q working scheme**

1. To identify the objects and their sub sets - by "learning" on "query" basis.
2. To create agent objects and mediator objects with their corresponding entity functions and properties
3. Issue service routines to act on objects
4. Generate policy requests for OSQ and handling replies from agent objects.
5. Update MQ-Graph
6. Select and establish path for transfer

**Steps followed in media transfer**

Procedure for "learning" required QoS parameters

If Agl, Ag2 are agent objects of a call originator (host / server) and call receiver (client).

C1 = Call_Agent (Agl, BW, HIGH)
C2 = Call_Agent (Ag2, BW, NULL)
D1 = Call_Agent (Agl, Trff, HIGH)
D2 = Call_Agent (Ag2, Trff, HIGH)
E1 = Call_Agent (Agl, TimDelay, HIGH)
E2 = Call_Agent (Ag2, TimDelay, HIGH)

Procedure for "Querying" agent for Negotiation

On_Query ( ( C1 > C2 ) and ( D1 > D2 ) and ( E1 > E2 ) )

Select the parameters on Condition
1. **CPU Agent** (c1, c2, c3)
   
   Where c1 = Available Memory,  
   c2 = Processor Speed of system,  
   c3 = Video Accelerator Card Configuration

2. **Memory Agent** (b1, b2)
   
   Where b1 = Buffer allocation  
   b2 = Virtual Memory

Required QoS parameter values:

3. **VOD Agent** (d1, d2, d3, d4)
   
   Where d1 = Bandwidth (Mbps)  
   d2 = Expected Time Interval for play out (secs)  
   d3 = Maximum Delay (secs)  
   d4 = Frames Created per sec

4. **Net Agent** (n1, n2, n3)
   
   Where n1 = Available bandwidth  
   n2 = Frame Transfer per sec  
   n3 = Buffer Capacity

---

**Agent Handling**

Check up procedures - check for bandwidth, media packets, traffic intensity for a media streaming application

BWav = Available bandwidth  
BWrq = Required bandwidth  
BWcl = Bandwidth at Client node

BWcl = BWav

If (Query (agent1 (BWav, MP, TRF)) AND (agent; (BWav, MP)))  
AND (agent1 (BWav, TRF)) = agentcl (BWrq, TRFrq, MPrq)

If (Ag1(A1), Ag2 (A1), Ag3(A1) ) AND

If (Ag1 (BW1), Ag2 (Trff1))
.
.
.
.
.
.
End If

The calculation is based on data gathered from an active agent interactivity among caller and callee is shown in Figure-6.9.
Fig-6.9 Interactivity of Me-ADO-Q procedures during a call session

Filter Properties:
Number of Channels Assigned  - 2
Number of Samples per sec  - 44100
Average no of bytes played  - 176400

Stream Characteristics:
Stream Parameters  - 44100 Hz
Bit Transfer Rate  - 128K Bit/s
Frames Decoded  - frames
Sync Lost  - 1 time
Block Size  - 1 Byte
Frames per second  - 20 Frames

Stream Characteristics
On enroute for the call receiver Me-ADO-Q system dynamically create agents as each new components are being identified with corresponding set of parameters.

6.8 Case Study and Result
This section presents on evaluation of the Me-ADO-Q model comprising of MQ algorithm for implementing run-time support architecture,
Evaluation and testing is done on complex media streaming application, Video On Demand implemented on Linux.

![Fig 6.10 Me-ADO-Q Traffic Management System Architecture](image)

The experiment can also be executed on clients running on heterogeneous platforms (e.g. Linux and Windows) with varying network bandwidth or CPU availability (e.g. Pentium III-based notebook PC). Such setup requires intermediate proxies running at gateway agent servers to trans-code the media to different formats, e.g. H.261 or uncompressed bitmap. E-Model has been setup for various media streaming network types, by simulation using test traffic. Within the media application, critical software components include the following:

a. The MPEG streaming service, located on the server

b. The players that play video either with MPEG, H.261 or bitmap formats, depending on the bandwidth or buffer availability of each client

c. The tracking filter that implements a collection of computationally intensive tracking algorithms, which processes incoming video and outputs the coordinates of the tracked object

d. The transcoders, which perform MPEG-to-H.261 or MPEG-to-bitmap transcoding.
6.8.1 Implementation

Run-time architecture is used to monitor and adapt a complex Video On Demand System. The application is developed using CORBA based client/server architecture running on Linux Operating System. The server has various capacities of on-line and off-line video shows, which are to be distributed to various client systems based on their requirement. The clients can be heterogeneous portable devices like Palm OS based PDAs, Wireless LAN based systems, Ethernet based LAN running on different OS platforms. This application provides good facility for the user to select any video clippings of their choice, which may vary for different time intervals. Both on-line and off-line video clippings are to be demonstrated over the net. It should be understood that in any video streaming application it is important to keep the end-to-end delay below 100 ms between the camera source and the video window screen as a destination for real time online system [16] or between the server and clients.

The model uses Me-TRAP-Q scheme as Traffic Management module for testing purposes. Different Entity Agents and their corresponding Component Agents are created. We provide Entity Agents E1, E2, E3, E4 with various Media Player - Application Agent A1, Operating System Agent A2, Network Agent A3 and other low level agents to inter-co-operate together. (Fig-6.8, Fig-6.9 and Fig-6.10).

<table>
<thead>
<tr>
<th>Entity Agent</th>
<th>Components</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>E1</strong></td>
<td>A1, A2, A3</td>
</tr>
<tr>
<td><strong>A1</strong></td>
<td>MPEG streaming service on the server with Windows NT as OS</td>
</tr>
<tr>
<td><strong>A2</strong></td>
<td>Buffer to accommodate the frames, tracking frame unit</td>
</tr>
<tr>
<td><strong>A3</strong></td>
<td>MPEG player on the client.</td>
</tr>
<tr>
<td><strong>E1</strong> is suitable for a high performance client, which has sufficient CPU, bandwidth, and energy resources.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Entity Configuration</th>
<th>Components</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>E2</strong></td>
<td>B1, B2, B3, B4</td>
</tr>
</tbody>
</table>
B1 - MPEG streaming service on the server with LINUX as OS
B2 - MPEG-to-H.261 transcoder on the proxy, and tracking filter
B3 - H.261 player on the client.
B4 - Buffer on Hold of 120bps for play-out or transfer

E2 is suitable for a client with sufficient CPU and energy resources, but without a high-speed network connection.

Entity Configuration E3 Network Proxy Agent consists of three application components Cl, C2, C3
C1 - Ethernet Network of 100Mbps speed.
C2 - Varying Transmission Traffic Intensity of > 40 Mbps and < 70Mbps
C3 - Bitmap player on the client.

Entity Configuration E4 Network Proxy Agent consists of three application components
D1 - Play-out Memory of 170bps.
D2 - Varying Transmission Traffic Intensity > 40 Mbps and < 70Mbps
D3 - MPEG player on the client.

Fig-6.11 shows agents created for Video-On-Demand application in clients and server systems. The interactivity among objects is also shown in Figure, where each agent communicate with other agents using Policy Manager to negotiate and deliver OSQ value among each agent. If any agent differs in OSQ value then an alternate agent with expected OSQ value is searched on ResourceLearning( ) and ResourceDiscovery( ) procedures to obtain the required agent with OSQ.

Fig 6.11 Me-ADO-Q Model for Video On Demand Application
6.8.2 Results and Discussion

Fig-6.12 shows the packet loss, play-out delay and end-to-end obtained for streaming video on demand application. Results are shown for identifying end-to-end delay and percentage of media packet loss for various schemes embedded with Me-ADO-Q model.

a) End-To-End Delay -
Me-TRAP-Q shown in Table-6.2 exhibits an average of 29ms of play-out delay, similar to Me-TRAP-Q scheme in Chapter -V. While end-to-end delay is found as 107ms in average that is best as to the existing schemes [5,6,7,15,23,29,41,99] discussed earlier. Me-TRAP-Q scheme is replaced with MeTrffSchl scheme and DiffServ Scheme. MeTrffSchl scheme with Me-ADO-Q model shown in Table-6.3 shows an average of 110ms to 124ms end-to-end delay, while play-out delay on average is found to be 32ms, which minimizes jitter and hence better quality of service. On experimenting with DiffServ scheme, the model shown in Table-6.4 exhibits surprising results as compared to normal DiffServ scheme, end-to-end delay is found to be varying from 168ms to 172ms, which is reduced compared to normal DiffServ approach, which exhibits an average of 186ms.

b) Media Packet Loss:
Percentage of media packet loss in Me-TRAP-Q scheme with Me-ADO-Q approach is 1.03% during worst bursty traffic conditions. With Me-TrffSchl scheme media packet loss is found to be 1.10% similar to actual Me-TrffSchl scheme, and with DiffServ scheme percentage of packet loss varies from 1.87% to 1.93%. The result is slightly higher than Me-PLM scheme and Me-TRAP-Q scheme, since media packets are not given higher priority.
On experimenting Me-ADO-Q with none of schemes, by default it works with Best Effort approach. The result in Table-6.5 shows that packet loss is reduced by 30%. Of all the schemes used with Me-ADO-Q scheme it is suggested that Me-TRAP-Q scheme provides a better solution.

c) QoS Metric:

E-Model based QoS metric provides metric as quality for streaming media transferred over IP network using various schemes with Me-ADO-Q model. Me-TRAP-Q scheme and Me-TrffSchl scheme with Me-ADO-Q model is assigned value 4.6, which is agreed as Best Quality in worst-case situations. While value 4.75 is assigned during normal conditions. Me-PLM scheme carries value 3.75, since it controls media packet loss and not control end-to-end delay nor manages traffic for providing end-to-end quality. Table-6.6 and Fig-6.15 shows the MOS metrics against corresponding schemes.

6.9 Summary

Me-ADO-Q addressed the problem of handling end-to-end QoS provisioning in continuous media transmission in broadband networks. Effective self-healing method Me-ADO-Q – CATALYST, diagnoses and eradicates the chance of an unexpected media quality failure. A working scheme was being developed to analyze and regulate media flow transfer on variable parameter setups. Finally, validation of the effectiveness of model and policy schemes through a set of simulations and experiments prove effectiveness of the model. The model is found to provide an optimistic end-to-end QoS solution than other QoS methods like DiffServ, which has been used, for comparison in results with existing schemes.
Table-6.2 : Me-ADO-Q Model with Me-TRAP-Q Scheme

<table>
<thead>
<tr>
<th>Sl. No</th>
<th>Sender Address</th>
<th>Receiver Address</th>
<th>Average Number of Bytes Send</th>
<th>Average Traffic Intensity at Gateway / Routers</th>
<th>Me-TRAP-Q</th>
<th>Me-ADO-Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MAT Systems Coimbatore</td>
<td>MAT Systems Denver</td>
<td>106,230</td>
<td>317,424</td>
<td>12</td>
<td>110</td>
</tr>
<tr>
<td>2</td>
<td>MAT Systems Sunnyvale</td>
<td>MAT Systems Coimbatore</td>
<td>110,346</td>
<td>417,144</td>
<td>16</td>
<td>112</td>
</tr>
<tr>
<td>3</td>
<td>MAT Systems Denver</td>
<td>Hisco Bangalore</td>
<td>106,130</td>
<td>365,412</td>
<td>10</td>
<td>111</td>
</tr>
<tr>
<td>4</td>
<td>Hisco Bangalore</td>
<td>MAT Systems Sunnyvale</td>
<td>107,020</td>
<td>378,744</td>
<td>12</td>
<td>115</td>
</tr>
<tr>
<td>5</td>
<td>MAT Systems Coimbatore</td>
<td>Hisco Bangalore</td>
<td>106,763</td>
<td>248,064</td>
<td>5</td>
<td>114</td>
</tr>
<tr>
<td>6</td>
<td>MAT Systems Denver</td>
<td>MAT Systems Sunnyvale</td>
<td>107,618</td>
<td>309,396</td>
<td>0</td>
<td>103</td>
</tr>
</tbody>
</table>

Average number of bytes lost (bytes)
- Me-TRAP-Q scheme : 9.163
- Me-ADO-Q with Me-TRAP-Q : 7.104

Average End-to-End Delay (ms)
- Me-TRAP-Q scheme : 110.83
- Me-ADO-Q with Me-TRAP-Q : 107.65
Comparison Delay of Me-TRAP-Q with Me-ADO-Q Model

Fig-6.12 Comparison of End-to-end Delay time of Me-TRAP-Q Scheme with Me-ADO-Q Model
Table 6.3: Me-ADO-Q Model with Me-TrffSchl Scheme

<table>
<thead>
<tr>
<th>Sl. No</th>
<th>Sender Address</th>
<th>Receiver Address</th>
<th>Average Number of Bytes Send</th>
<th>Average Traffic Intensity at Gateway / Routers</th>
<th>Me-TrffSchl</th>
<th>Me-ADO-Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MAT Systems Coimbatore</td>
<td>MAT Systems Denver</td>
<td>106,230</td>
<td>317,424</td>
<td>20</td>
<td>117</td>
</tr>
<tr>
<td>2</td>
<td>MAT Systems Sunnyvale</td>
<td>MAT Systems Coimbatore</td>
<td>110,346</td>
<td>417,144</td>
<td>26</td>
<td>112</td>
</tr>
<tr>
<td>3</td>
<td>MAT Systems Denver</td>
<td>Hisco Bangalore</td>
<td>106,130</td>
<td>365,412</td>
<td>10</td>
<td>117</td>
</tr>
<tr>
<td>4</td>
<td>Hisco Bangalore</td>
<td>MAT Systems Sunnyvale</td>
<td>107,020</td>
<td>378,744</td>
<td>12</td>
<td>115</td>
</tr>
<tr>
<td>5</td>
<td>MAT Systems Coimbatore</td>
<td>Hisco Bangalore</td>
<td>106,763</td>
<td>248,064</td>
<td>5</td>
<td>114</td>
</tr>
<tr>
<td>6</td>
<td>MAT Systems Denver</td>
<td>MAT Systems Sunnyvale</td>
<td>107,618</td>
<td>309,396</td>
<td>0</td>
<td>110</td>
</tr>
</tbody>
</table>

Average number of bytes lost (bytes)
- Me-TrffSchl scheme: 12.16
- Me-ADO-Q with Me-TrffSchl: 8.05

Average End-to-End Delay (ms)
- Me-TrffSchl scheme: 114.13
- Me-ADO-Q with Me-TrffSchl: 110.06
Comparison of Delay of Me-TrffSchl Scheme with Me-ADO-Q Model

Fig-6.13 Comparison of End-to-end Delay results of Me-TrffSchl with Me-ADO-Q (Me-TrffSchl) Model
Table-6.4 : Me-ADO-Q Model tested with DiffServ Scheme

<table>
<thead>
<tr>
<th>Sl. No</th>
<th>Sender Address</th>
<th>Receiver Address</th>
<th>Average Number of Bytes Send</th>
<th>DiffServ</th>
<th>Me-ADO-Q</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Average Number of Bytes Lost</td>
<td>Delay (ms)</td>
<td>Average Number of Bytes Lost</td>
</tr>
<tr>
<td>1</td>
<td>MAT Systems Coimbatore</td>
<td>MAT Systems Denver</td>
<td>106,220</td>
<td>10</td>
<td>115</td>
</tr>
<tr>
<td>2</td>
<td>MAT Systems Sunnyvale</td>
<td>MAT Systems Coimbatore</td>
<td>110,345</td>
<td>15</td>
<td>110</td>
</tr>
<tr>
<td>3</td>
<td>MAT Systems Denver</td>
<td>Hisco Bangalore</td>
<td>106,125</td>
<td>12</td>
<td>105</td>
</tr>
<tr>
<td>4</td>
<td>Hisco Bangalore</td>
<td>MAT Systems Sunnyvale</td>
<td>107,129</td>
<td>13</td>
<td>126</td>
</tr>
<tr>
<td>5</td>
<td>MAT Systems Coimbatore</td>
<td>Hisco Bangalore</td>
<td>106,760</td>
<td>9</td>
<td>127</td>
</tr>
<tr>
<td>6</td>
<td>MAT Systems Denver</td>
<td>MAT Systems Sunnyvale</td>
<td>107,600</td>
<td>0</td>
<td>103</td>
</tr>
</tbody>
</table>

Average number of bytes lost
- DiffServ scheme : 9.30
- Me-ADO-Q with DiffServ : 7.32

Average End-to-End Delay
- DiffServ scheme : 114.33
- Me-ADO-Q with DiffServ : 108.51
Table-6.5 Me-ADO-Q tested with Best Effort Scheme

<table>
<thead>
<tr>
<th>SI No.</th>
<th>Sender Address</th>
<th>Receiver Address</th>
<th>Average Number of Bytes Send</th>
<th>Number of Bytes Received</th>
<th>Average Traffic Intensity at Dishnet Gateway / Routers</th>
<th>Best Effort Delay (ms)</th>
<th>Me-ADO-Q Delay (ms)</th>
<th>Average Number of Bytes Lost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MAT Systems Coimbatore</td>
<td>MAT Systems Denver</td>
<td>106,224</td>
<td>106,112</td>
<td>317,424</td>
<td>112</td>
<td>210</td>
<td>80</td>
</tr>
<tr>
<td>2</td>
<td>MAT Systems Sunnyvale</td>
<td>MAT Systems Coimbatore</td>
<td>110,344</td>
<td>110,168</td>
<td>417,144</td>
<td>176</td>
<td>173</td>
<td>102</td>
</tr>
<tr>
<td>3</td>
<td>MAT Systems Denver</td>
<td>Hisco Bangalore</td>
<td>106,136</td>
<td>106,040</td>
<td>365,412</td>
<td>96</td>
<td>216</td>
<td>40</td>
</tr>
<tr>
<td>4</td>
<td>Hisco Bangalore</td>
<td>MAT Systems Sunnyvale</td>
<td>107,024</td>
<td>106,904</td>
<td>378,744</td>
<td>120</td>
<td>196</td>
<td>87</td>
</tr>
<tr>
<td>5</td>
<td>MAT Systems Coimbatore</td>
<td>Hisco Bangalore</td>
<td>106,768</td>
<td>106,720</td>
<td>248,064</td>
<td>48</td>
<td>167</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>MAT Systems Denver</td>
<td>MAT Systems Sunnyvale</td>
<td>107,608</td>
<td>107,576</td>
<td>309,396</td>
<td>32</td>
<td>203</td>
<td>4</td>
</tr>
</tbody>
</table>

Average number of bytes lost
- Best-Effort Scheme : 17.34 bytes
- Me-ADO-Q Model : 13.53 bytes

Average End-to-End Delay
- Best-Effort Scheme : 194.15 ms
- Me-ADO-Q Model : 164.50 ms
Table 6.6: E-Model QoS metric for various schemes under Me-ADO-Q Model

<table>
<thead>
<tr>
<th>Schemes Tested</th>
<th>% of Media Packets Lost</th>
<th>End-to-end Delay</th>
<th>Play-out Delay</th>
<th>E-Model MOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Me-TRAP-Q</td>
<td>1.03</td>
<td>107ms</td>
<td>29ms</td>
<td>4.6 (Best)</td>
</tr>
<tr>
<td>Me-TrffSchl</td>
<td>1.10</td>
<td>110ms</td>
<td>32ms</td>
<td>4.4 (Best)</td>
</tr>
<tr>
<td>Me-PLM</td>
<td>1.02</td>
<td>181ms</td>
<td>66ms</td>
<td>3.7 (Fair)</td>
</tr>
<tr>
<td>Diff-Serv</td>
<td>1.87</td>
<td>168ms</td>
<td>54ms</td>
<td>4.1 (Good)</td>
</tr>
<tr>
<td>Best-Effort</td>
<td>3.20</td>
<td>194ms</td>
<td>90ms</td>
<td>2.8 (Aver)</td>
</tr>
</tbody>
</table>

Fig 6.14: E-Model based QoS metric for schemes using Me-ADO-Q