An Adaptable Quality of Service for Multimedia on the Internet

Gerard Parr, Kevin Curran
Telecommunications & Distributed Systems Research Group
Northern Ireland Knowledge Engineering Laboratory
University of Ulster, Coleraine Campus, Northern Ireland, UK
Email: **kj.curran@ulst.ac.uk**

There are two general approaches to providing for isochronous streams in the current Internet. The first approach is the resource reservation approach through protocols such as RSVP, or ATM technology. This provides bandwidth guarantees, however, it also requires significant upgrading of resources in the underlying network. The other common approach is adaptive rate control where the end-system has control of its rate according to feedback from the client population. This approach cannot guarantee timely delivery and raises some scaling questions, however a properly implemented scheme does improve quality and it requires no changes to the underlying IP network. Hence, there exists a dichotomy of requirements; 1. To cater for reservation protocols or 'hooks' for future reservation components, and 2. To provide an architecture which provides an application controlled QoS scheme, which scales to the size of the current Internet in a best-effort architecture.

We have built a pure Java architecture named Chameleon which caters for the requirements above. We cater for the resource reservation camp by providing an adaptive framework with hooks for interchanging QoS modules at run-time. We provide this service through the use of the Reflection API of Java. With reflection, the QoS modules may be developed from the outset as meta objects thus allowing the systems functional requirements to be separated from the non-functional (QoS) requirements thus allowing future QoS modules to be incorporated in a safer and more cost efficient manner.

We cater for the rate adaptive group by distinguishing between the various media components (e.g. video, audio) providing separately composed protocol stacks for each of the media which deliver the media to separate quality (high, medium and low) multicast group addresses allowing heterogeneous clients to connect according to their resources. Multicast groups overcome the scaling problem and clients subscribe to each group in accordance with resource availability and move between groups according to bandwidth availability.

Our goal is to provide an architecture that is appropriate for the environment of the Internet (long latencies, high message loss rates, relatively frequent network partitioning) with scalability to both large numbers of nodes and distances.

1 Introduction

The use of audio-visual media on the Internet continues to grow. Products such as Real Audio [1] and IPCast[2] Video viewer are becoming increasingly common. However, multimedia due to its timely nature requires guarantees different in nature with regards to delivery of data from TCP traffic for ordinary HTTP requests.

In this paper we present a multimedia middleware enabling architecture, called Chameleon (so named because it changes in harmony with the environment), that delivers audio and video on a best-effort network. It supports the heterogeneous client population of the Internet by providing various qualities of media and sending these media to separate multicast groups so that clients can 'pick and choose' media groups in accordance with available resources.

Resource reservation protocols, pricing schemes etc. are just some of the capabilities that are required by clients. To cater for this, we provide future clients with the necessary 'plug-points' for them to 'plug' in their components at run-time. Reflection allows us to separate the functional requirements from the non-functional requirements such as resource reservation protocol modules through use of metaobject protocols.

Chameleon is guided by a desire to create an architecture that can evolve and adapt to an ever-changing environment similar to living things in their struggle for survival. We wish to impart this quality into our middleware, as we believe good software architectures are all about being adaptable and resilient to change.

2 Approaches to QoS on the Internet

QoS control in distributed systems is a current research area due to the emerging new types of multimedia streaming applications requiring real time delivery of information and an Internet integrated services architecture

[1] www.realaudio.com
[2] www.ipcast.com

enabling the sharing of the network between data streams with different QoS requirements (real-time or best effort). QoS management allows the real time delivery of information as it encompasses both the control and delivery of QoS in an integrated services architecture.

Networking technologies such as ATM have various categories of guarantees ranging from best-effort (attempts to reach destination – not guaranteed) to guaranteed (system has failed if it fails to deliver all data). However, the Internet is composed of a heterogeneity of networking architectures offering varying services, therefore, we find that the guarantee of multimedia applications such as video conferencing is an impractical on a network such as the Internet at present.

Another proposal for applying QoS control on the Internet is the emerging QoS control protocol being defined by the Internet Engineering Task Force (IETF): the Resource reSerVation Protocol (RSVP) [1]. RSVP is being defined as part of a new effort for enabling integrated service on the Internet: the ISA (Integrated Service Architecture). RSVP is based on flow control and receiver initiated reservation by setting priorities in modified routers along the data path on the Internet. It uses a set of messages to establish, maintain and release reservations. It is still in an experimental phase at the time of this project.

In general, these mechanisms have focussed on regulating competition for network resources among traffic sources at the network level. They involve various aspects in resource reservation and allocation, as well as flow control, admission control and negotiation protocols used by cell, packet and transport layers.

2.1 Application Controlled QoS

The research direction that we have followed is in the area of introducing adaptation behaviour into the application at the end-system, (and throughout the middleware). This is general referred to as Application Controlled QoS, beyond the control mechanisms of the transport layer. This perspective is different from the resource reservation approach in that it mainly deals with protocols and techniques located and operated in the middleware architecture and end-systems, rather than intermediate network switches. The advantage of this approach over the previous one is that it does not need to radically modify the existing protocols already implemented and running in current networks, so that the QoS delivery could be implemented with least modifications.

2.2 A Centrally Allocated Mechanism

With application controlled QoS, the adaptation behaviour that is currently designed for the applications is usually in built into the application such as the video player or an audio player and the multimedia playout control algorithm is implemented inside each application, so that the application can maintain a certain expected playout level for the client when adapting to fluctuations in the delivery of media.

There is an obvious disadvantage to this approach in that various systems are working individually on their various adaptive algorithms, yet, they are incapable of collaborating to provide an overall integrated service to the user. Each application may respond differently to network bottlenecks. Some may consume a greater share of resources at a critical stage in another applications computation, while other applications may choose to ignore system perturbations.

Hence, a design consideration of our architecture is to provide a central resource allocation/synchronisation unit which controls the adaptation behaviour of each application and balance the resources required during the adaptation.

3 CHAMELEON

Chameleon is a programming abstraction, which supports dissemination oriented communication. The abstraction is analogous to that of the various broadcast media in everyday use such as newspaper, radio and TV corresponding to text, audio and video components contained in multimedia applications. People, like the various qualities of both and may listen to the radio while reading a newspaper etc. Likewise, Chameleon fragments the various media elements of a multimedia application and ‘broadcasts’ them over separate channels to be subscribed to at the receiver’s own choice. We do not force the full range of media on any subscriber, a source simply transmits onto a specific channel, and receivers, which have subscribed to that channel, receive media streams (e.g. audio, text and video) without explicit interactions with the source.

Chameleon builds upon the Java middleware software iBus [2], which allows the stacks to be created dynamically and provides a group communications paradigm which enables us to deal with clients according to developer specified ‘relatedness’ such as IP domains for bandwidth allocation decisions in a principled manner.

The Chameleon communication paradigm is a major departure from more traditional approaches, in that a source and a set of receivers are very loosely coupled in their control and data exchange interactions. In general, the source's main concern is to push various media streams onto a channel, without emphasis on where they end up (i.e. who the actual receivers are), and how they are used (i.e. what specific receivers will extract from any or all of the streams). A receiver's main concern is what to extract from a channel, which is viewed as offering multiple streams, some or all of which are of interest. We believe this communication paradigm is appropriate for multimedia distribution services required by a large class of multimedia multipoint applications. A prime example is 'video distribution' [3] as in cable television systems where a single source generates video (and associated audio) distributed to a large set of receivers who generally have little or no interaction with the source.

Chameleon differs from the current distributed multimedia frameworks in that we separate the media into text, audio and video and distribute these to multicast groups. We also create multiple groups within each category e.g. high, medium and low audio quality thereby allowing receivers to subscribe to groups such as high audio with medium video and text or simply low audio. This is where our work is unique. No work that we are aware of has being conducted in this area where the various components of multimedia are separated and transported across the network according to their characteristics. We address the network congestion and heterogeneity problem by taking into account the differing nature and requirements of multimedia elements such as text, audio and video thereby creating tailored protocol stacks which distribute the information to different multicast groups allowing the receivers to decide which multicast group(s) to subscribe to according to processing power and network bandwidth availability.

The primary goal of Chameleon is to provide an infrastructure for building multimedia computing platforms that support interactive multimedia applications dealing with synchronised time-based media in a heterogeneous distributed environment. Systems exist which support interactive applications, which deal with synchronised time-based media. Most existing systems however, have not been written with reuse in mind, thus, the major focus of Chameleon in this paper is the ability to support insertion/removal of modules concerned with the applications non-functional QoS requirements.

3.1 Dynamically Composable Stacks

Chameleon is motivated by the need for a responsive robust middleware and the desire to provide optimal quality of service for multimedia to a heterogeneous client population on a large-scale network such as the Internet. Chameleon achieves this by recognising that multimedia is composed of audio/video and text components. These components can have different qualities and can require different qualities of service. Text, for example, may need to be received by each client without loss, therefore the text component is sent through a reliable (non-loss) channel whereas video may be sent through a best-effort (lossy) streaming channel. Multicast groups provide for a large-scale client population without the single server bottleneck.

A protocol stack consists of a linear list of protocol objects and represents a quality of service such as reliable delivery, virtual synchrony, or encrypted communication. The framework provides the services necessary for supporting new communication protocols and qualities of service. Chameleon consists of a set of Java classes for representing Uniform Resource Locators, protocol stacks, the framework API and posting objects. Dynamically composable protocol stacks overcome the limitations imposed by generic protocol stacks allowing optimisation for particular traffic.

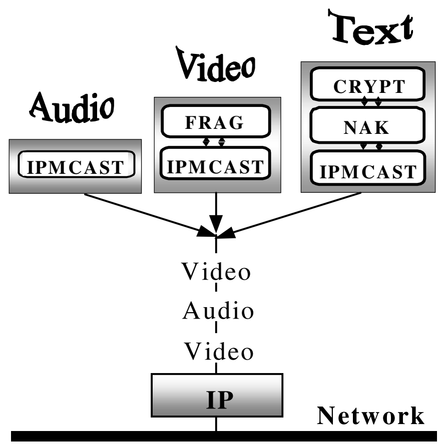

Figure 1: Dynamically composable stacks

The Chameleon paradigm is configured as follows. Media files (audio/video/text and synchronisation parameters) are stored on a server as high, medium or low QoS stacks. These are transported through separately optimally configured stacks via multiple multicast groups to heterogeneous receivers. A receiver may upgrade or downgrade within each stack, according to fluctuating conditions in the network. A component called a Session Director sits at a well know address and has full knowledge of the various QoS stacks each server utilises to ship the media to various multicast groups. Clients connect to the Session Director to find the multicast group locations and the quality of service parameters for the particular media stacks. Chameleon monitors each receivers quality and the system continually aims for the optimal throughput in accordance with client resource capabilities. There is an obvious trade-off decision to be made between opting for reconfiguration and the benefits to be gained from each new QoS. We believe that framework must be adaptable so as to evolve with the organisation.

3.2 Reflection

Reflection allows us to separate the functional requirements of an application (what it does) from the non-functional ones (how it does it). It is based on the Meta-object Protocol (MOP) defined in 1987 by Maes in [4]. The Java Reflection API puts particular emphasis on the use of Reflection in distributed systems by taking advantage of the network-centric capabilities of Java. Reflection is the base of our model and is evaluated in this project.

Reflection in Java 1.1 refers to the ability of Java classes to reflect upon themselves, or to "look inside themselves". The java.lang.Class class has been greatly enhanced in Java 1.1. It now includes methods that return the fields, methods, and constructors defined by a class. These items are returned as objects of type Field, Method, and Constructor, respectively. These new classes are part of the new java.lang.reflect package, and they each provide methods to obtain complete information about the field, method, or constructor they represent, however, it still falls short of providing the ability to incorporate run-time behavioural reflection necessary for inserting modules at run-time.

Dalang [5] is an extension for Java that introduces behavioural runtime reflection. It uses class wrappers to implement a simple metaobject protocol where a metaobject (in the form of a wrapper) controls the behaviour of a base level object. The metaobject can redefine the handling of method calls in order to add desirable properties such as fault tolerance or security. Standard introspection can be achieved by using the standard java.lang.reflect package. These wrappers can be added dynamically or statically. We used this class library to enable us to insert modules for increased performance at run-time with no change necessary to the Java Virtual Machine.

For example, we may use the Method object which has methods to query the name, the parameter types, and the return type of the method it represents to allow us to inspect the members of each protocol stack class to check if it supports a relevant protocol profile that is optimal for the current network conditions. Our reflective implementation also allows us to form the meta-circular tower of components. Hence, we may have replication objects calling QoS objects calling logging objects. It is a goal of our system to be monitoring routes from to different multicast groups for each of the media and choose the route most efficient over a period of time. We hope to apply reflection in this area. The insertion of a RSVP module or pricing module or security module (such as described in [6]) could be achieved in this manner.

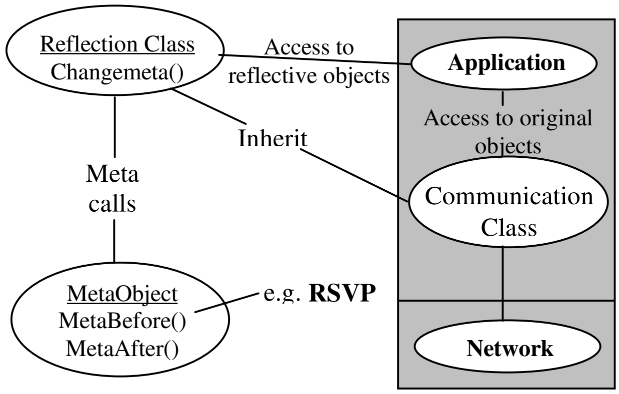

Figure 2: Redirecting a call to a metaobject

The insertion of modules is transparent to the application. Figure 2 illustrates this in the fact that when the application sends data to the communication class, it is reflected to the metaobject protocol which calls the RSVP module, thus performing resource reservation management functions and then onto the network as if no redirection had occurred.

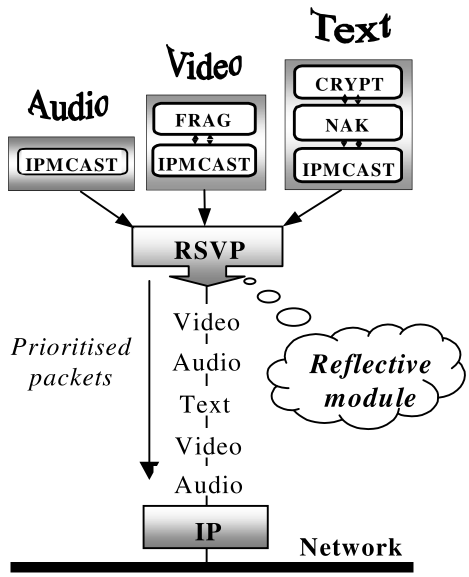

Figure 3: Run-time insertion of Reservation module

Prioritisation of packets among other functions can be performed by the insertion of ‘filter’ modules in the pipeline at run-time as illustrated in Figure 3. It is through the reflection API that Chameleon achieves the dynamic reconfigurable facilities to cater for varying network conditions.

3.3 Central Resource Synchronisation Unit

As mentioned earlier, it is desirable to provide a central resource allocation unit which controls the adaptation of each application so that the sum of the parts work as a cohesive whole in providing QoS to the user.

As Chameleon, is concerned with the transport of media elements to the end-user which may differ over time in that a user may choose to retrieve audio only, or sound and vision or sound, audio and sub-titles, therefore, some synchronisation facility which adapts to the users choices must be in place.

The upper layer that we currently implement in the layered framework is the Java Media Framework API. This is the media API developed by the creators of Java to handle multimedia.

To enable us to create an architecture which would allow the user to drop media and add media during playout required particular rules to be implemented to enable this. One rule we decided upon was to make the audio, the master player so that video and sub-titles were synchronised to the audio playout. This we felt was the most natural means of keeping the media in synchrony.

4 Insertion of Authentication and Tariff Policies Example

Chameleon is primarily concerned with shipping various media quality text, audio and video files to heterogeneous clients through various multicast groups. The standard system allows a client to connect to the Session Director, specify the various media qualities it requires and begin retrieval. There is no authentication and no tariffs. This is illustrated in Figure 4.

However, consider the case where a media server might wish to ensure that only subscribed members may retrieve media files and that these media files are subject to various pricing tariffs.

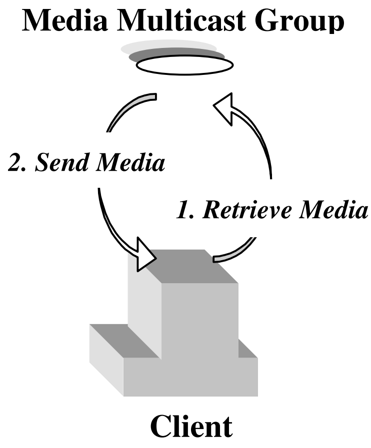

Figure 4: Standard Media Retrieval Policy

The technique employed by Chameleon is as follows. The user establishes a connection to a security service sitting at a well-known address port broadcast by the Session Director. The client specifies the address of the media server it wishes to connect to. The authentication service then consults pricing policies and access policies. It then grants privileges according to rights and sets charges. The user may then proceed. The protocol described in Figure 5 follows these steps.

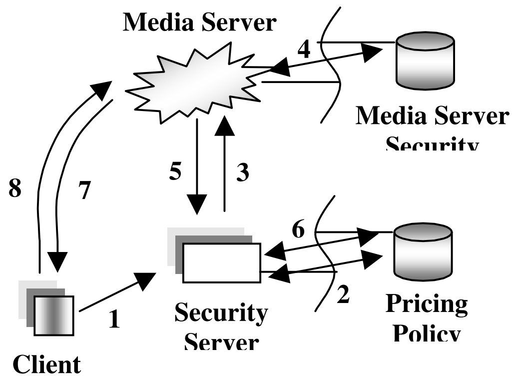

Figure 5: Upgrade of service - security with pricing

1. The remote client connects to the security server.
2. The security server authenticates the remote user (by verifying user name, password and destination domain and checks pricing policy).
3. The security server connects to the desired media server.
4. The media server confirms acceptance of the call.
5. The security server logs acceptance.
6. The media server authenticates the remote user and accepts or rejects the tunnel.
7. The media server exchanges information with the client for the duration of the session.

8. The client exchanges data with the server using the session id for the duration of the session.
9. Upon termination by one party or other, the cost of the session is forwarded to the pricing policy server and logged.

The architecture also allows a pricing only (no security) policy to be configured or a security only policy such as described in [9]. All this is achievable at run-time through the reflective hooks provided by Chameleon's reflective API.

5 Conclusion

There exists no existing guaranteed means of transporting multimedia end-to-end in the Internet at present. There are a host of technologies offering to fill this gap including the next IP version (IPV6) to RSVP. These services require upgrade of routers throughout the global network and thus seem unlikely to come to fruition in the near future.
Chameleon has sought to provide a principled means of allowing a system to be adapted throughout its lifetime with the minimum of effort.

Reflective programming provides a principled means of planning for and coping with change in a computing system. The Internet at present with its multiple standards and the interconnection of components such as decoders, middleware, synchronisation scripts, databases, QoS modules requires more than a plug, try and play mentality [8].

Chameleon is basically a pure Java layered-framework incorporating a multimedia synchronisation framework (JMF) with a multicast group communication framework (iBus) with a reflective architecture (Dalang) to cater for change in transporting multimedia on a best-effort network. We used the scenario of a security and pricing policy being incorporated into Chameleon at run-time. The system may never have seen either that particular security or pricing component before. There is no overhead for ordinary method calls in our system except when the call is routed to a meta-object or a tower of meta-objects. We expect to apply reflective methods in many areas of our middleware in coping with dynamic adaptation.

References

1. Zhange, L., Deering, S., Estrin, D., Shenker, S. RSVP: a new resource reservation protocol. IEEE Network, Vol. 7, No. 5, pp. 8-18, September 19

2. Maffeis, Silvano. The iBus software bus. http://www.softwired.ch. Online Documentation, 1999

3. Sincoskie, W.D. System architecture for a large-scale video-on-demand service. Computer Networks and ISDN Systems. North Holland, Vol. 22, No. 2, 1991

4. Maes, P. Concepts and experiments in computational reflection. OOPSLA'97, 1997

5. Welch, Ian and Stroud, R. Using metaobject protocols to adapt third party components. Middleware '98, Lancaster, 1998

6. Kumar, V.P. Beyond Best Effort: Router architectures for differentiated services of tomorrows Internet. IEEE Communications magazine. Vol. 36, No 5, May 1998

7. Java Media Framework Specification. Sun Microsystems, 1997. . www.javasoft.com

8. Clark, D.D, Tennenhouse, D.L. Architectural considerations for a new generation of protocols. Proc. ACM SIGCOMM'90, pp. 200-208, September 1990

9. Parr, Gerard and Curran, Kevin. Optimal multimedia transport on the Internet. Journal of Network and Computer Applications, Vol. 21, pp. 149-61, 1998.