A Framework for the Development of Tolerant Real Time Applications

Pedro Sousa <pns@uminho.pt>
Vasco Freitas <vf@uminho.pt>

Departamento de Informática
Universidade do Minho
P-4709 Braga, Portugal

Keywords: RTP, RTCP, WWW, Plug-ins, Best effort services.

Abstract

This work presents a framework architecture for the development of distributed real-time applications to be integrated into WWW clients. It assumes a WWW environment over networks providing a best-effort delivery service like the internets based on the IP protocol.

The framework is that of an application programming interface (API) providing the program developer with the services needed by tolerant real-time applications. Once developed, an application is bundled together with the API to form a WWW plug-in which can subsequently be called from a WWW client interface or browser. The application is then perceived as being integrated into the WWW environment.

The design aims to provide real-time applications with a transport service layer assuring near end-to-end isochronism despite the weak guaranties of the underlying network service. The implementation of the mechanisms that allow multi-stream real-time communications to adapt to the operational conditions of these networks are discussed. In this work, the RTP and RTCP protocols were also implemented as part of the API.

Experience with this framework reports the development of a prototype real-time application for multimedia group communication and the analysis of the behaviour of RTP sessions in a real operational situation. The analysis uses protocol state data logged during their operation.

1 Introduction

Many applications can be found nowadays in network environments, some of which, by their nature, try to draw quite heavily from the available network resources. Real-time applications fall into this category as they often require high data flow rates, end-to-end isochronism, limits on delay and jitter, stream synchronisation and so forth, leading to operational paradigms not matched by the underlying networks. The IETF (*Internet Engineering Task Force*) has produced some work in this area [1, 4]. The main problem is the nature of the IP protocol, which only provides for a *best-effort* service although new ideas are being put forward namely the Resource Reservation Protocol (RSVP) [2, 25]. In such networking environments it is up to the applications themselves to devise and implement the mechanisms needed to satisfy their requirements. This work, therefore, is concerned and proposes a programming framework providing a developer of real-time applications with the support needed by the requirements of this class of applications. The following section will briefly review some of the related background. In section 3 the functionality of the proposed framework is described. Section 4 reports on the experience in developing a prototype application and on the analysis of the behaviour of RTP sessions in operational situations. Section 5 draws some conclusions and indicates further directions that are being followed up.

2 Background

Classes of aplications. Two kinds of applications can be found in the Internet scenario, *real-time* and *elastic* applications [1, 4]. Elastic applications, such as Telnet, FTP or E-Mail are not delay sensitive which does not mean that users will not suffer the consequences of a larger variability of delay or insufficient bandwith, resulting in a poorer quality of service. Real time applications on the other hand are delay sensitive.

Real-time applications can be divided in two diferent classes: *rigid* and *tolerant*. Rigid applications require a type of service that can assure limited delay variation, packet loss, etc. Tolerant applications must adapt their behaviour to the operational conditions of the underlying networks and protocols and have the ability to overcome problematic network conditions. They assume a network service with no performance figures.

Adequacy of protocols. In the case of the Internet stack of protocols, TCP [5] is a connection oriented protocol providing a reliable end-to-end data transmission service. It is a complex protocol implementing various mechanisms to ensure among other things that data is not lost. This is an essential property in transactions data communications but not in many others of the kind of real-time conferencing communications where TCP is not the most apropriate transport service. On the other hand, UDP [15] although a non reliable datagram protocol adding very little to the underlying IP network protocol, has the strong feature of being well suited to multipoint communications [7]. UDP has been the choice of the transport service for the proposed framework.

Previous work. The quest for running real-time audio over the Internet was perhaps the main initial motivation behind the studies of real-time applications over packet data networks [13, 16]. Most of the work in the field is credited to Schulzrinne [17, 18, 19]. The interest rapidly spread to other media like real-time video or moving pictures. New protocol proposals have come up [20, 21, 22, 12] which have been tested in some applications. A well known case is the use of the MBONE (Multicast backBONE) [23, 8] to run multicast audio and video conferencing over the wide area, having RTP as the underlying protocol.

Target applications. The programming interface that has been developed and will be presented here is targeted at the following types of candidate applications:

- Applications which generate traffic with specific real-time characteristics.
- Applications for group communications that use multicast.
- Applications that are able to use transport layer signalling to adapt themselves to changing network conditions and strive for optimal performance.
- Real-time applications that have been developed for the WWW.

3 Description of the framework

The framework is that of an application programming interface (API) providing the program developer with the services needed by tolerant real-time applications, such as, remote unicast and multicast connections, communication rate adaptation, playout buffering, synchronisation of real-time data streams, jitter control as well as user interface control. The API implements the Real Time Protocol (RTP) and its akin protocol, the Real Time Control Protocol (RTCP), which make use of the host system UDP transport service to establish peer communications with replicated versions of the application, eg, over IP networks. Once developed, an application is bundled together with the API to form a WWW plug-in which can subsequently be called from a WWW client interface or browser. The application is then perceived as being integrated into the WWW environment.

The typical scenario envisaged in this work is that of a WWW application for multimedia group communication which is registered at some Web server. Figure 1 depicts this setup. A remote

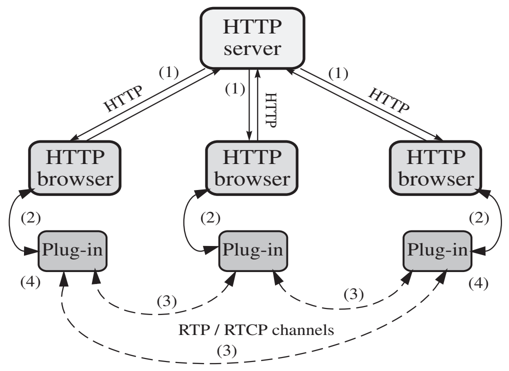


Fig 1: The operational scenario

user who wishes to join a session over this application does so by selecting it on his WWW client, figure 1 (1). A WWW *plug-in*, fig. 1 (4), will then be activated on that user client which takes over control of his participation in the session, figure 1 (2). It opens RTP and RTCP channels to the other clients in the session, as shown in figure 1 (3), and from that moment on the user is able to exchange multimedia data with the session participants.

This work concerns the protocol layer that are embeded in the *plug-ins*, which is made available to the user in the form of an Application Programming Interface (API). The following sections discuss this protocol layer.

3.1 Architecture of plug-in

Protocol components of API. Figure 2 shows the underlying protocol components of the API. It is assumed that IP is the network protocol over which is used UDP as the transport protocol

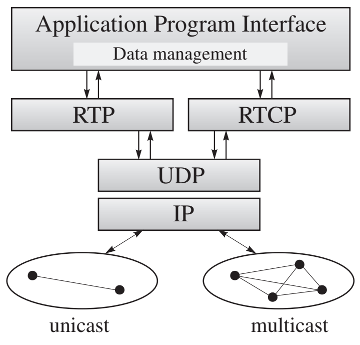


Fig 2: Protocol components of API

for the Real Time Protocol, RTP, and the associated Real Time Control Protocol, RTCP. Over this *real time session layer* sits the application interface which uses the services specified for that layer and provides applications with a set of specified programming functions.

Layer model of plug-in. Plug-ins comprise the following five modules which are depicted in figure 3:

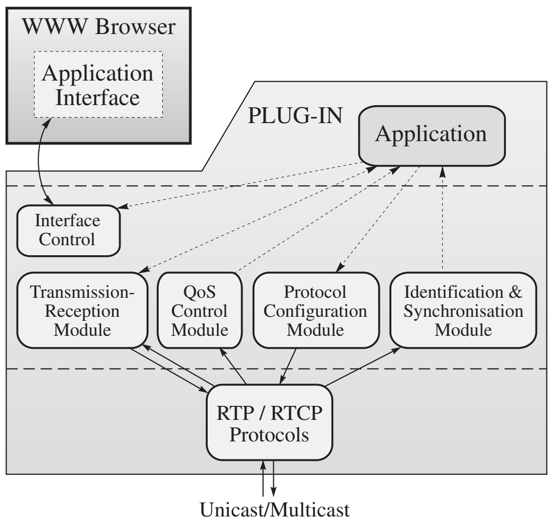


Fig 3: Layer model of plug-in

1. Interface management module.
2. Transmission and reception module.
3. QoS control module.
4. Identification and synchronization module.
5. Protocol configuration module.

Each of these components are discussed in the following sections.

3.1.1 Interface management

As previously mentioned the aim is to integrate the whole interface underlying the user application into the WWW client. Hence, the interface management module will allow the programmer to handle a given area of its client interface. To achieve this, a capability of WWW plug-ins was used together with the HTML command *embed* that allows to specify a given area in the client interface which an application can handle and becomes responsible for. In this case the application is the plug-in proper that implements the application. The user application can afterwards address that same area to perform any operation involving, for example, text handling and graphics.

3.1.2 Transmission and reception

This module performs all operations involving passing data (messages) to and receiving data from the transport service layer, to be forwarded to and arriving from the network concerning all stations participating in the session. During reception this module constantly updates the RTP state variables, such as sequence numbers, jitter values, packet loss, and so on, and passes the data on to the application. In its reception rôle, the module also implements a packet colection procedure similiar to the algorithm proposed in the anexes to RFC1889 [20], which takes into account the acceptable delays in receiving out of sequence packets.

3.1.3 QoS control

The main rôle of the QoS module is to monitor the status of RTP connections and, from observed parameters, to provide the application with the means to adapt itself to the short term forthcoming (expected) conditions. The application is then able to react to packet loss, jitter and other network loading effects. The techniques used involve the handling of damping buffers.

Packet delay and jitter. A factor of crucial importance in real time applications is delay. More important still is jitter, the time variation in delay, as applications need to maintain end-to-end isochronism. For each transmitting protocol entity, each receiving entity keeps an estimate of jitter from past observed values. The expected jitter is computed from the single exponential smoothing forecast[1] [11]

$$J_n = \alpha X_n + (1-\alpha) J_{n-1} \qquad (1)$$

[1]A property of single exponential smoothing is that the weight of previous values decreases exponentially.

where J_n is the current forecast value of jitter, X_n is the current *measured* jitter value, J_{n-1} is the previous forecast jitter value and $\alpha \in [0,1]$ is a real constant, whose value is chosen according to the desired influence of the new measured value on the forecast. The measurement of X_n is made indirectly from the timestamps on the RTP data packets. Let τ_n be the timestamp imposed upon packet n at the moment it leaves the sender and t_n the timestamp for the same packet recorded at the moment it reaches the receiver on a given RTP channel. The transit time of packet n computed from

$$\delta_n = t_n - \tau_n \tag{2}$$

is taken as the channel delay[2] at time t_n and it is a measure of network delay at this instant. The current *measured* jitter in formula (1) is taken as

$$X_n \;=\; \delta_n \;-\; \delta_{n-1} \tag{3}$$

Transmmitter and receiver clocks need not be synchronised.

Playout buffering. Playout buffering has been used to smooth out delay and thus reduce jitter. A *playout buffer* is a damping memory used to hold packets for a variable but controlled amount of time before the destination application actually use them. Most real time applications allow for a certain amount of delay without a noticeable loss of quality. Up to one hundred miliseconds delay in interactive speech and video is still quite acceptable. To compensate for jitter, therefore, the application should know the maximum delay value of any packet within a session. Let Δ be this value. Then, for each packet arriving with a computed delay of δ_i, the application would only *play* the packet after $\Delta - \delta_i$ units of time have elapsed, during which time it would remain in the playout buffer. As Δ cannot be known in advance, a *local* value is derived from a sample of the delays of the first m arriving packets (for which no compensation is performed). Let

$$\delta_{n-m}, \delta_{n-m+1}, \delta_{n-m+2}, \cdots, \delta_{n-2}, \delta_{n-1} \tag{4}$$

be the sequence of the delays of the m packets that have arrived immediately before t_n and δ_{I_k} the kth larger value in sequence (4). If a function $\Delta(m,k)$ of differences in delay is defined as follows

$$\Delta(m,k) \;=\; \delta_{I_{k+1}} \;-\; \delta_{I_m} \tag{5}$$

then the value $\Delta(m,0)$ is taken as an estimate of Δ for $t_i \geq t_n$ and is passed on to the application. Δ is recalculated and updated periodically. Additionally, it can be seen that playout buffering implicitely performs packet ordering.

[2]it is a mean value over a time interval of δ_n units of time ending at instant t_n.

Packet loss. Receivers periodically report to transmitters about packet loss thus allowing transmitters to compute the current packet loss rate. Upper and lower loss rate thresholds are defined at the transmitters so that whenever the actual loss rate either exceeds the upper threshold or goes below the lower threshold, the transmitter either falls back or raises up to a more appropriate transmission rate in an attempt to maintain the quality of the transmission. The actual loss rate value used is a filtered version of the loss rate computed at each instant. The filtering function used is, again, the single exponential smoothing forecast filter as in equation (1)

$$FL_n \;=\; \lambda\, L_n \;+\; (1-\lambda)\, FL_{n-1} \tag{6}$$

where FL_n is the current filtered loss rate value, L_n is the current loss rate value computed from the reported packets lost, FL_{n-1} is the previous filtered loss rate value and $\lambda \in [0,1]$ is a real constant suitably chosen. Figure 4 displays actual values from our experiments, collected from a unicast RTP channel, showing channel throughput increasing initially as the (filtered) packet loss rate lies below the lower threshold, which has been set at 10%, then reaching a value such that the loss rate overshoots the upper threshold, which has been set at 15%, thereby forcing the transmission rate to decrease until the loss rate returns to within limits. Figure 4 also shows that

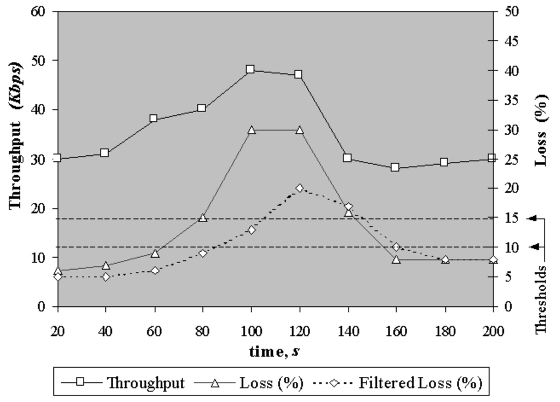


Fig 4: Transmitter adapts to network conditions

in this particular situation the *controlled variable* (loss rate), swings between above and below the *locking zone* defined by the threshold limits. Although this is the expected behaviour of this control system, it may eventually become unstable, in that oscillations may indefinitely increase in amplitude with time and possibly contribute to periodically driving the network to a congestion state. Parameter λ is the tunning knob on the feedback path of this system and its value must be carefully chosen as the system is non-linear and certainly a time-variant one as the underlying network is being loaded by other independent and variable traffic flows.

A transmitter classifies each receiver as being in one of three states: *congested*, *normal* or *minimal loss* according to whether the value of its filtered packet loss rate lies above, between or below thresholds *(b)* and *(a)* shown in figure 5. In a

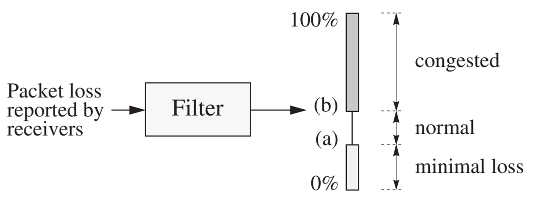


Fig 5: Receiver classification dynamic [3]

multicast situation receivers may be in different states and therefore a decision as to whether to increase, maintain or decrease the transmission rate is not so simple to take as in unicast. The approach suggested by Busse *et al* [3] was adopted whereby, basically, if the number of receivers in the congested state does not exceed a *very small* proportion P_d of the total number of receivers, the transmitter maintains the current transmission rate and it only increases the rate if, additionally, the number of receivers in the minimal loss state are in excess of a *reasonable* proportion P_m. Transmission rate is only decreased if the number of congested receivers raises beyond P_d. Let N_c and N_n be the number of receivers in the *congested* state and in the *normal* state, respectively. If N_t is the total number of receivers in the multicast session, then

$$\begin{array}{lll} \text{if} & \dfrac{N_c}{N_t} \geq P_d & \text{then} \quad \text{decision} \leftarrow \text{decrease} \\ \text{elseif} & \dfrac{N_n}{N_t} \geq P_m & \text{then} \quad \text{decision} \leftarrow \text{maintain} \\ & & \text{else} \quad \text{decision} \leftarrow \text{increase} \end{array}$$

3.1.4 Identification & synchronisation

Identification. This module enables applications to access high level information associated with each of the session participants. Each participant makes available through the RTCP channel metainformation on itself.

Synchronisation. When a session involves the transmission of more than one *medium* in separate channels, such as in a video-phone session with two RTP channels, one for video and the other for voice, these need to be synchronised, ie, aligned in time, as they may be subject to different delays. Applications dealing with multimedia streams are kept informed on the status of synchronism of the various media, ie, whether one stream is ahead of another and for how much so that the application can take the necessary measures to bring them to alignment, eg, by dropping out packets from the other stream and shorten its delay in the playout buffer as depicted in figure 6. This functionality uses information exchanged

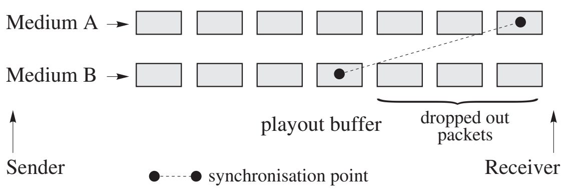


Fig 6: Alignment of multimedia streams

through the RTCP channels namely the transmitter timestamps for each of the media streams.

3.1.5 Protocol configuration

This module takes care of the necessary parameter setting before a multicast session may begin. There are two types of such parameters: network and application. The former concern for example multicast addresses, protocol ports and so on. The latter relate to the initialisation of values of constants and variables in the programs that implement the functionalities refered to previously, eg, constants for filters, threshold values for the QoS controller and for the receiver classifier, etc.

3.2 Programming API

The protocols in the plug-in API were developed in C++. A hierarchy of objects was defined which implements the functionalities of the RTP and RTCP protocols. The programming interface at the disposal of an application developer takes the form of a programming object (*App_Session*) which includes a set of methods to which the programmer has access. Within this object, one or more RTP/RTCP sessions may coexist which have been started and may be killed by the programmer. Figure 7 depicts the struture of this object. By way of example, some methods within this object are the following:

App_Session(), ∼App_Session() Class constructor and destructor.

Add_Session(sess_param,func_param,ident_param, adjust_param) Method to add a new RTP session.

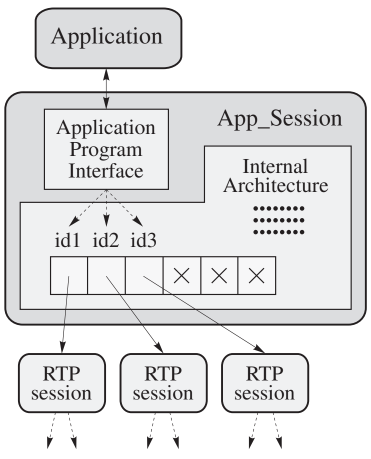


Fig 7: Struture of object *App_Session*

Takes the protocol configuration structures as parameters.

Session_number() A function returning the number of active RTP sessions.

Get_decision(id) A function that examines the states of all receivers and returns the decision value to be taken as to the transmission rate of the transmitter.

Number_entities(id) A function returning the number of participants in a RTP session.

Send(id,apt,tam,marker) A function that sends a RTP packet to the specified session.

Get_max_buffer(id,ssrc) A function returning an estimate of the playout buffer size to be used for the specified source within the specified session.

Synchronise(id1,id2,name) The specified channels are to be synchronised.

Get_reception_state(id,ssrc) For the specified session and source, returns a structure containing statistics on how packets are being received from that source up to the current moment.

Get_source_id(id,ssrc) For the specified session and source, returns a structure containing high level information about the source.

List_entities(id) For the specified session returns a list of the participants identifiers.

4 Experience

A prototype application of a generic platform for multimedia group communication has been developed under this framework. The implementation includes tools for state data collection enabling the runtime behaviour of real time applications and the underlying mechanisms to be analysed. These tools are quite useful in that they allow to monitor and evaluate the effects of different network loads on the system performance.

4.1 An application for multimedia group communication

The implementation consists of an experimental cooperative application which makes use of the API and the underlying system whose components have been described in the preceeding sections. It has the following features:

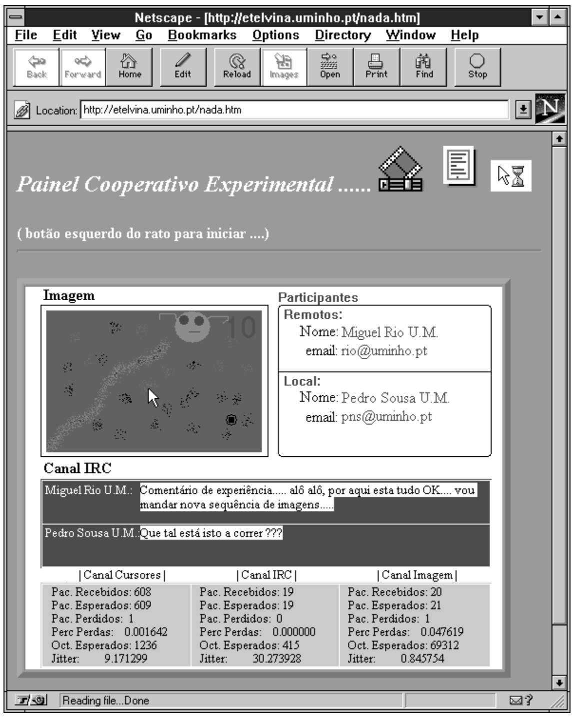
- An *image* transmission channel which may be accessed by all prospective participants.
- A *cursor-position* transmission channel for all participants.
- A *message* transmission channel (IRC[3]).
- A *directory-board* channel identifying all current participants.
- A *channels-status* area displaying the status of all active channels.
- It constitutes a fully integrated plug-in to a WWW client and uses the set of layered protocols described.

Figure 8 shows a WWW client (browser) interface displaying on the large (white background) window a session of cooperative work with two participants using this application. On the upper right hand corner of this window is the directory-board display identifying the two participants by their name and e-mail. The upper left hand corner displays the shared image and the cursors from all participants, from their respective channels. On the full width middle strip area of this window is located the shared IRC message display area with one sub-area dedicated to each participant and finally on the full width bottom area lies the channels status display area showing, in this case, raw parameter values for channels cursor-position, IRC-message and image.

These channels correspond to autonomous RTP sessions which participants join when they initiate their application thus joining the overall cooperative session. Control information of each of those (sub)sessions flow in the associated RTCP channels. All participants are identified by some metainformation on the directory-board channel which is derived from the data they plublicise in the matching RTCP channel, namely, in the source description packets[4].

[3]Internet Remote Chat

[4]SDES packets


Netscape - [http://etelvina.uminho.pt/nada.htm]
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Edit Reload Images Open Print Find Stop
Location: http://etelvina.uminho.pt/nada.htm

Painel Cooperativo Experimental ......

(botão esquerdo do rato para iniciar ....)

Imagem

Participantes
Remotos:
Nome: Miguel Rio U.M.
email: rio@uminho.pt
Local:
Nome: Pedro Sousa U.M.
email: pns@uminho.pt

Canal IRC
Miguel Rio U.M.: Comentário de experiência..... alô alô, por aqui esta tudo OK.... vou mandar nova sequência de imagens.....
Pedro Sousa U.M.: Que tal está isto a correr ???

|Canal Cursores|	|Canal IRC|	|Canal Imagem|
Pac. Recebidos: 608	Pac. Recebidos: 19	Pac. Recebidos: 20
Pac. Esperados: 609	Pac. Esperados: 19	Pac. Esperados: 21
Pac. Perdidos: 1	Pac. Perdidos: 0	Pac. Perdidos: 1
Perc Perdas: 0.001642	Perc Perdas: 0.000000	Perc Perdas: 0.047619
Oct. Esperados: 1236	Oct. Esperados: 415	Oct. Esperados: 69312
Jitter: 9.171299	Jitter: 30.273928	Jitter: 0.845754

Reading file...Done

Fig 8: A session of multimedia cooperative work

The image channel: session RTP_1. In the image channel flow streams of coded still or moving images to all participants. It is a prototype of a channel where video data sourced at one or more participants flows in real-time to all the other participants. In this prototype GIF89a [6] was used for coding images. This format is mainly used for still images although it may easily be adapted to coding of moving images [24] despite its low performance as compared to MPEG and other compressors.

The cursor channel: session RTP_2. Remote signalling of cursor position of a participant is implemented by a call to a routine whenever his pointing device moves. This routine has access to the cursor coordinates which are then transmitted over this channel to all participants.

The messages channel: session RTP_3. This is a discussion channel in writen message medium and it is an approach to a communication channel using IRC. The medium is broadcast to all participant interfaces and displayed in separate and identified display areas.

4.2 Analysis of application runtime behaviour

Tools in the programming interface have been used to collect performance data on the runtime behaviour of real time applications and the underlying framework. The API and the protocol software developed may be run in debug mode in which specified timestamped state variable values are logged onto a file. Log data may then be cross checked against logs of the other participants and analysed.

A case study. Two simultaneous RTP sessions were setup on each of the three hosts, A, B and C which were connected to a 10BASE2 ethernet LAN. In a first scenario each participant host generated 1 Kbyte packets per RTP session at approximately 40 ms intervals (25 packets/s), ie, an average agregate load on the LAN of 1.2 Mbps. In a second scenario the load was doubled to approximately 2.4 Mbps by halving the interpacket times generated by each RTP session at each host (50 packets/s).

For each of the two scenarios, values of reported packet loss and packet buffering time for all three participants were recorded. Figures 9 and 11 plot four consecutive values of these two variables for each participant after approximately 12 seconds had elapsed, when transmissions had stabilized.

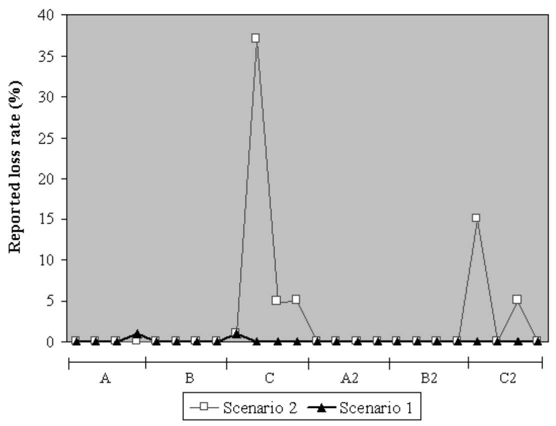


Fig 9: Reported packet loss in the two scenarios

The values for each participant are identified as A, B and C. For each scenario a second run was performed under the same conditions giving values identified as A2, B2 and C2 on the plots.

Packet loss. In both scenarios packet loss rate was very small except for that reported by participant C in the increased load scenario where it early started to increase as shown by the first two values of both C and C2 in figure 9. This event was explained by the fact that machine C had the poorer hardware of all three, ie, less resources and of a lower performance.

Rate adaptation. At some instant when packet loss rate reported by C exceeded the upper threshold, loss rate control started to act at hosts A and B which decreased their transmission rate on the reported channel. Its effect can be seen in the third and fourth values of both C and C2 in figure 9 where loss rate decreases back to a value near zero. The effect can also be seen in figure 10 which plots total packet loss from host A observed at host C, on the same channel. Total

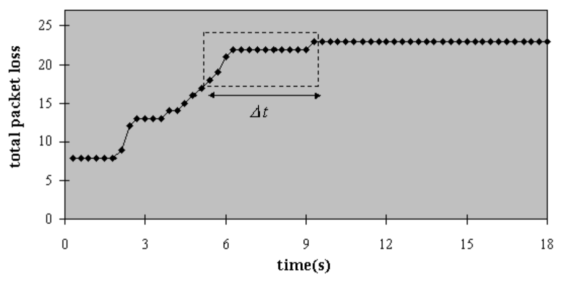


Fig 10: Packets lost from A observed at C

packet loss increases initially but it stabilises at a near constant value some Δt seconds after the control action has started.

Playout buffering. Queueing time at the playout buffers become more variable in the second scenario as shown in figure 11. One explana-

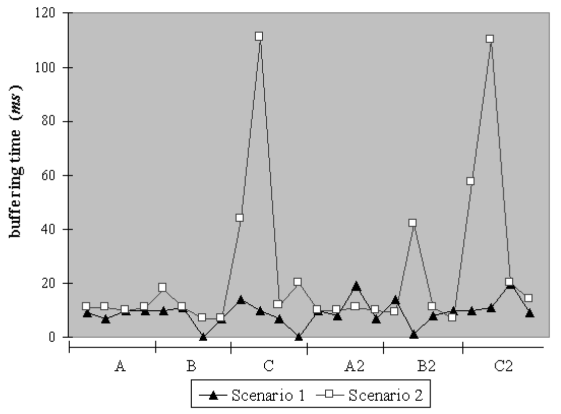


Fig 11: Packet buffering time in the two scenarios

tion for this phenomenon is that a higher load caused not only by higher transmission rates but also because stations increase the rate almost synchronously, are likely to increase jitter as a consequence of harsher conditions in packet admission into the network where increased contention may introduce more variable delays at the physical interface.

Stream alignment. The two RTP streams are to be synchronised. Figure 12 plots the higher load scenario shifts in alignment (synchronisation) between the two streams, as recorded at host B, relative to their transmission from A. Alignment is lost quite frequently and its shift

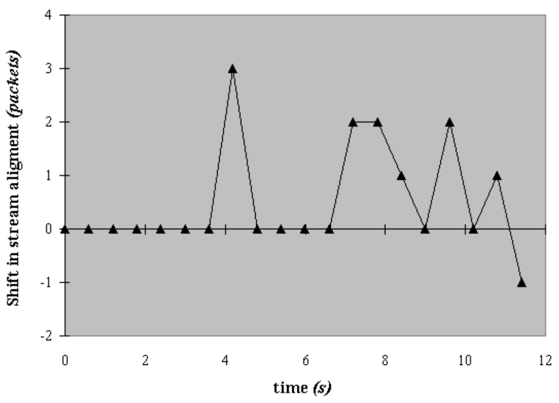


Fig 12: Shift in stream alignment at B

is clearly seen to oscillate around zero or thereabout, thus indicating that it is being successfully controlled. At the lower load scenario, both streams stay well aligned. Shifts are rare and barely noticed amounting to no more than one packet.

5 Conclusions and further work

The concept of developing tolerant real-time multimedia applications as plug-ins for the WWW client interface seems to be a reasonably good integrating concept. The proposed framework is that of an API layer providing the programmer of this class of applications with the type of services they need when the underlying network only assures a best-effort connectionless service, namely:

- user interface control
- remote unicast and multicast connections
- communication rate adaptation
- playout buffering
- alignment of real-time streams
- jitter control

This conclusion is backed up by experience where a complete prototype distributed application for multimedia group communication over a local network was developed and run on a working LAN. The behaviour of the application and the mechanisms that provide the services were checked from operational logs, which the system also provides, and proved to perform reliably and reasonably well as far as could be expected from the available resources and the somewhat demanding conditions of the application.

Some aspects of this system are currently being tackled which will eventually lead to a more robust and efficient platform. They are:

- Compression of RTP headers. As in certain uses of the TCP/IP protocols [10], there have been proposals for compressing RTP headers [14].
- Interoperability with multicast notification and management mechanisms, and session register and announcement [9].
- Consideration of the use of mixers.
- Coexistence of adaptive and resource reservation protocols such as RSVP [2, 25]

Acknowledgment

The first author was sponsored by scholarship BM/6667 from PRAXIS XXI.

References

[1] R. Branden, D. Clark and S. Shenker. *Integrated Services in the Internet Architecture: an Overview.* RFC1633, Jul 1994.

[2] R. Braden, L. Zhang *et al.* *Resource Reservation Protocol (RSVP) - Version 1: Functional Description.* Internet draft, IETF, Nov 1996.

[3] I. Busse, B. Deffner and H. Schulzrinne. *Dynamic QoS Control of Multimedia Applications based on RTP*, May 1995. gaia.cs.umass.edu/pub/Buss9601/Dynamic.ps.gz

[4] D. D. Clark, S. Shenker and L. Zhang. *Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism.* Internet draft, IETF, Oct 1996.

[5] D. E. Comer. *Internetworking With TCP/IP, Volume I: Principles, Protocols, and Architecture.* Prentice-Hall, 2nd ed.

[6] Compuserve Incorporated. *Graphics Interchange Format(sm), version 89a*, Jul 1990.

[7] C. Diot, W. Dabbous and J. Crowcroft. *Multipoint communications: A survey of Protocols, Functions, and Mechanisms.* IEEE Journal on Selected Areas in Communication, (15)3, Apr 1997.

[8] H. Eriksson. *MBONE: The Multicast BackBone.* Comm of the ACM, (37)8, Aug 1994.

[9] M. Handley and Van Jacobson. *SDP: Session Description Protocol.* Internet draft, IETF, Nov 1995.

[10] V. Jacobson. *Compressing TCP/IP Headers for Low-Speed Serial Links.* RFC1144, Feb 1990

[11] S. Makidrakis, S WeelWrigth, V. McGee. *Forecasting methods and applications.* John Wiley & Sons, pp 84-111, 1983.

[12] NACSE, Northwest Alliance for Computational Science and Engineering. *Conceptional and Historical Background of Real Time Protocols.* Dec 1996. //www.cs.orst.edu/~kleinro/nacse/rtp

[13] W. E. Naylor and L. Kleinrock. *Stream Traffic Communication in Packet Switched Networks: Destination Buffering Considerations.* IEEE Trans on Communications, (COM-30)12, pp 2534, 1982.

[14] S. Petrack. *Compression of Headers in RTP Streams.* Internet draft, IETF, Dec 1996

[15] J. Postel. *User Datagram Protocol.* RFC768, Aug 1980.

[16] L. Press. *Net.Speech: Desktop Audio Comes to the Net.* Comm of the ACM, (38)10, pp 25-31, Oct 1995.

[17] H. Schulzrinne. *When can we unplug the radio and telephone?.* NOSSDAV 95, 5th Int Workshop on Network and Operating Systems Support for Digital Audio and Video. Apr 1995. //gaia.cs.umass.edu/pub/schu9504/when.ps.gz

[18] H. Schulzrinne. *Internet Services: from electronic mail to real-time multimedia.* Proc of KIVS (Kommunication in Verteilten Systemen), pp 21-34, Feb 1995.

[19] H. Schulzrinne. *QoS for Real-time services: playout delay and application control.* Proc of the 46th RACE Concertation Meeting (RCM), Mar 1995.

[20] H. Schulzrinne, S. Casner, R. Frederick and V. Jacobson. *RTP: A Transport Protocol for Real-Time Applications.* RFC1889, 75 pp, GMD,..., XEROX, LBN Lab, Jan 1996.

[21] H. Schulzrinne. *RTP Profile for Audio and Video Conference with Minimal Control.* RFC1890, Jan 1996.

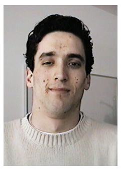
[22] H. Schulzrinne. *Some Frequently Asked Questions about RTP.* Sep 1996. //www.cs.columbia.edu/~hgs/rtp

[23] A. S. Thyagarajan, S. L. Casner and S. E. Deering. *Making the MBONE Real.* INET'95, May 1995. //www.ee.udel.edu/~ajit/inet95.dir/inet.html

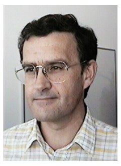
[24] K. H. Wolf. *Web Video - Interactive Video in the World Wide Web.* Proc 7th Joint European Networking Conf - JENC7, pp 1131-1137, Terena, Budapest, May 13-16, 1996.

[25] J. Wroclawski. *The Use of RSVP with IETF Integrated Services.* Internet draft, IETF, Oct 1996.

Vitæ



Pedro Sousa graduated in Systems and Informatics Engineering at the University of Minho, Portugal, in 1995 and was awarded a Masters degree in Informatics in 1997. He joined the Computer Communications group of the Department of Informatics in 1996 and has since collaborated in projects involving protocol development for real time networked applications.



Vasco Freitas is Associate Professor of Computer Communications at the University of Minho, Portugal. He graduated in electronic and telecommunications engineering in 1972 at the University of Lourenço Marques and received his M.Sc. and Ph.D. degrees from the University of Manchester (UK) in 1977 and 1980 respectively. From 1989 until 1994, he was in charge of the establishment and management of the Portuguese R&D Network (RCCN).