Buletinul Ştiinţific al Universităţii "Politehnica" din Timişoara

Seria ELECTRONICĂ şi TELECOMUNICAŢII
TRANSACTIONS on ELECTRONICS and COMMUNICATIONS

Tom 49(63), Fascicola 1-2, 2004

Developing VoIP Applications with QoS Support

Daniel Zinca[1], Virgil Dobrota[1], Aida-Virginia Gasparel[1], Virgil-Ioan Dragomir[1]

Abstract – This paper describes a VoIP application with Quality of Service support that was designed and implemented by the authors. The application runs under Microsoft Windows 2000/XP/2003 and includes the modules called Signalling, QoS, Media Transport and Voice. Whenever the IntServ approach (i.e. RSVP) could not be involved, an alternative solution based on DiffServ for IPv4/IPv6-based network applications is offered. The experiments using the proposed VoIP application were focused on PC-to-PC trials, as well as on PC-to-Phone configurations (through a Cisco 1750 router acting as a VoIP Gateway).

Keywords: VoIP, SIP, QoS, RSVP, DiffServ, NDIS Intermediate Drivers.

I. INTRODUCTION

VoIP (Voice over IP) applications are enabling voice communication over an IP network, typically over the Internet. The protocols involved are grouped in three categories, as in Fig.1: Signalling, Media Transport and QoS. For the media transport, the RTP (Real Time Protocol) is usually used, information being provided by voice codecs. For signalling the trend is to move towards IETF's SIP (Session Initiation Protocol), whilst the recent applications are investigating also the QoS aspects [1], [2].

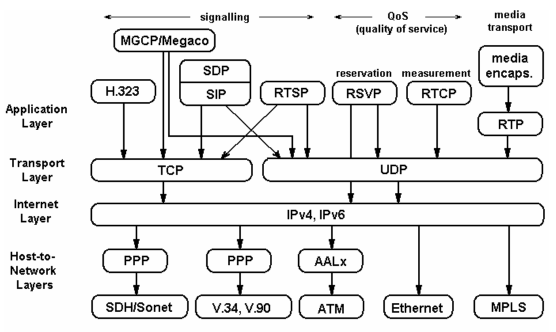


Fig.1. Protocols used by VoIP applications [3]

In this paper we present our results in designing and implementing a VoIP application, based on RTP, G.711 voice codecs, SIP 2.0 and IntServ solution for QoS (RSVP). Additionally, we developed a program that implements DiffServ on the terminal endpoint for applications that are not QoS-aware. Previously at ETc'2002 we presented the theoretical aspects, along with some practical solutions [4]. This time we were focused on the description of the applications, as well as on the experimental results. The paper is organized as follows: Section II describes the components of the VoIP application and Section III presents the DiffServ application. Next section is devoted to the experimental results and finally within Section V we draw the conclusions.

II. APPLICATION DESCRIPTION

The proposed VoIP application, running under Microsoft Windows environment, can act either as a client, i.e. the originator of the VoIP call, either as a server, i.e. the called part. Written in Microsoft Visual C++ 6.0, the application has the following modules:

- Voice Encapsulation
- Media Transport
- QoS (RSVP)
- Signalling (SIP)
- User Interface

The Voice Encapsulation Module is based on the Microsoft DirectX, Direct Sound technology. A primary streaming buffer is used and the *Play()* and *Stop()* methods are interacting with this buffer. The *Lock()* method must be involved before writing data into the buffer, whilst the *Unlock()* method is used before *Play()* can be called again.

The Media Transport Module consists on a class that implements the RTP Protocol and supports most of the payload types, including the Comfort Noise. The *SendRTPPacket()* method receives a pointer to the buffer filled with the voice samples and adds the header.

The RSVP Module uses Microsoft Windows Sockets 2.0 features to add QoS support to the RTP socket. This can be done after the two entities are negotiating the IP addresses and the ports to be used. The necessary parameters (`TokenRate, Latency, TokenBucket, DelayVariation`) are calculated based on the codec type. The function calls were described

[1] Technical University of Cluj-Napoca, Department of Communications, 26 Baritiu Street, 400027 Cluj-Napoca, Romania, Tel/Fax:+40-264-597083, E-mail: {Daniel.Zinca, Virgil.Dobrota}@com.utcluj.ro

in details in [4], the correct flow of RSVP messages being presented in Fig.2.

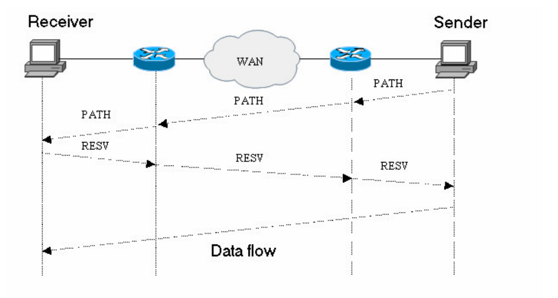


Fig.2. PATH and RESV RSVP's messages [5]

The Signalling Module, implementing SIP v2.0 and SDP protocols, contains a state machine and a message parser. Fig.3 presents the flow of SIP messages between two UAs (User Agents).

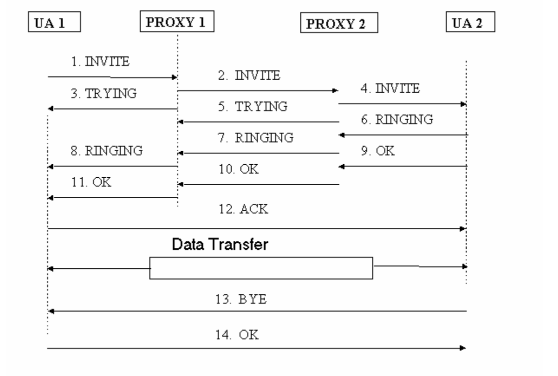
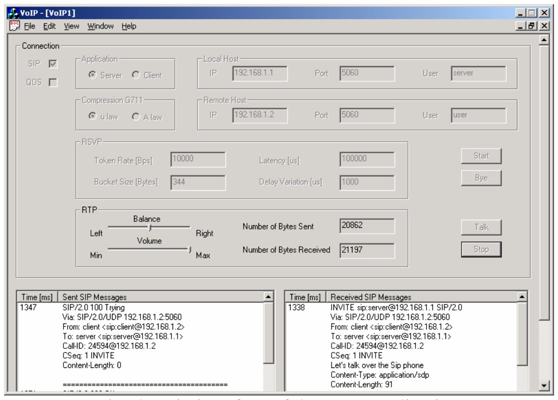


Fig.3 SIP messages between two User Agents

The last module is the User Interface, used to determine the user's requirements for the VoIP communication, as presented in Fig.4. Both A or µ laws can be selected for G.711 voice coding. Also QoS parameters can be changed and it can monitor the operation (number of bytes sent and received, SIP messages).



Connection

SIP ☑ QOS ☐

Application: [✓ Server] Client

Local Host: IP 192.168.1.1 Port 5060 User server

Compression G711: [✓ u law] A law

Remote Host: IP 192.168.1.2 Port 5060 User user

RSVP

Token Rate [Bps] 10000 Latency [us] 100000

Bucket Size [Bytes] 344 Delay Variation [us] 1000

RTP

Balance: Left — Right

Volume: Min — Max

Number of Bytes Sent 20862

Number of Bytes Received 21197

```
Time [ms]  Sent SIP Messages
1347       SIP/2.0 100 Trying
           Via: SIP/2.0/UDP 192.168.1.2:5060
           From: client <sip:client@192.168.1.2>
           To: server <sip:server@192.168.1.1>
           Call-ID: 24594@192.168.1.2
           CSeq: 1 INVITE
           Content-Length: 0

           =========================================
```

```
Time [ms]  Received SIP Messages
1338       INVITE sip:server@192.168.1.1 SIP/2.0
           Via: SIP/2.0/UDP 192.168.1.2:5060
           From: client <sip:client@192.168.1.2>
           To: server <sip:server@192.168.1.1>
           Call-ID: 24594@192.168.1.2
           CSeq: 1 INVITE
           Let's talk over the Sip phone
           Content-Type: application/sdp
           Content-Length: 91
```

Fig. 4 Main interface of the VoIP application

III. SUPPORTING DIFFSERV

A Differentiated Services architecture should "keep the forwarding path simple, push complexity to the edges of the network to the extent possible, provide a service that avoids assumptions about the type of traffic using it, employ an allocation policy that will be compatible with both long-term and short-term provisioning, and make it possible for the dominant Internet traffic model to remain best-effort" [6]. In the case of VoIP applications, EF (Expedited Forwarding) Service should be used because it provides low-loss, low-latency, low-jitter and assured bandwidth service. Because of the high importance of this PHB, it is critical that the forwarding behavior required and delivered by an EF-compliant node to be specific, quantifiable and unambiguous [7]. The packets marked with EF usually encounter short or empty queues, keeping also packet loss rate at minimum. Usually, traffic classification is made at the edge routers of the DiffServ architecture. Our solution enables traffic marking at the end-user computer, based on Layer 4 port used by the application.

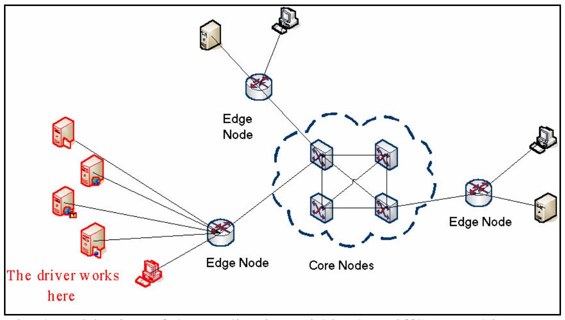


Fig.5 Positioning of the application within the DiffServ architecture

The following characteristics were implemented:

- support for IPv4 (modification of the ToS/ DSCP field within header) and IPv6 (modification of Traffic Class/ DSCP field)
- re-computation of the IPv4 header's checksum
- support for TCP and UDP Transport Layer protocols and identification of applications that are using certain ports.

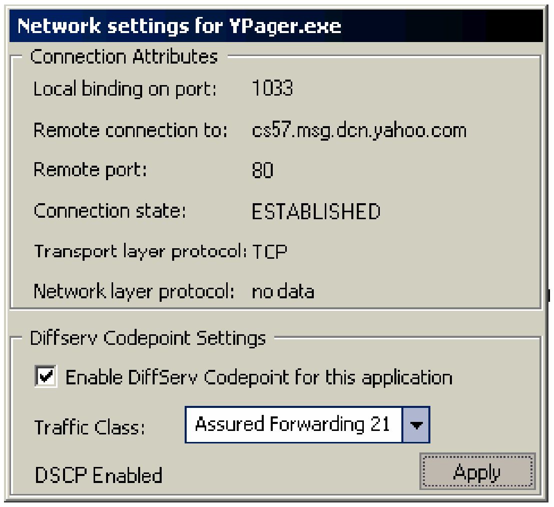
The application was developed using Microsoft Visual C++ .NET and Windows DDK environment, running in Windows 2000/XP/2003. The application has a complex structure, made out of several modules that interact synchronously (global classes) or asynchronously (Windows messages, global variables, global classes).

The actual core of the application performs the detection of Windows applications that initiate network communications, the management and the selection DiffServ-enabled programs, packet capture, driver communication handling.

However the core is masked by the graphical interface, which offers a very intuitive control over these commands, both by mouse and keyboard.

There were five modules: Graphical Module, Network Applications Detection Module, Driver Interface Module, Differentiated Services Module and Packet Capture Module.

The Graphical Module presents to the user all the details of the currently running applications that have network connections. There is one window interface where the applications using network services are listed and one auxiliary window that offers more details of the selected process. Fig. 6 presents the auxiliary window interface showing that the local administrator can enable DiffServ for a certain application and can select EF or other Traffic Class (in our example, AF21 is selected).


Network settings for YPager.exe

Connection Attributes

Local binding on port:	1033
Remote connection to:	cs57.msg.dcn.yahoo.com
Remote port:	80
Connection state:	ESTABLISHED
Transport layer protocol:	TCP
Network layer protocol:	no data

Diffserv Codepoint Settings

[✓] Enable DiffServ Codepoint for this application

Traffic Class: Assured Forwarding 21

DSCP Enabled

Fig.6 Enabling DiffServ and selecting AF21 Traffic Class within the Graphical Module

The list with all applications that are using TCP or UDP ports on the local machine is obtained by the Network Application Detection Module in two steps. First, by calling two functions present in *iphlpapi.dll* (*AllocateAndGetTcpExTableFromStack* and *AllocateAndGetUdpExTableFromStack*), the Process ID of each application is returned. By using it, two functions from *kernel32.dll* (*Process32First* and *Process32Next*) are returning the name of the corresponding application.

One important module is the Driver Interface Module, an NDIS 6.0 Intermediate Driver [4] that actually intercepts all IPv4 or IPv6 outgoing packets of the hosts. It checks for the proper port value, marks if appropriate the header and re-computes the checksum (last for IPv4 only). The Filter Intermediate Driver was chosen because it is layered between miniport drivers on the machine and the transport drivers. When a transport driver sends packets to a virtual adapter exposed by the intermediate driver, the second one reads the IP header, modifies if appropriate the DCSP field and propagates the packet to the underlying miniport driver. The handler used to perform all necessary operations to a packet is *MPSendPackets*.

The interaction between modules is presented in Fig.7. Note that 1 is the Packet Modification Module and 2 is the User-Mode Interface Module, both belonging to the DiffServ Driver.

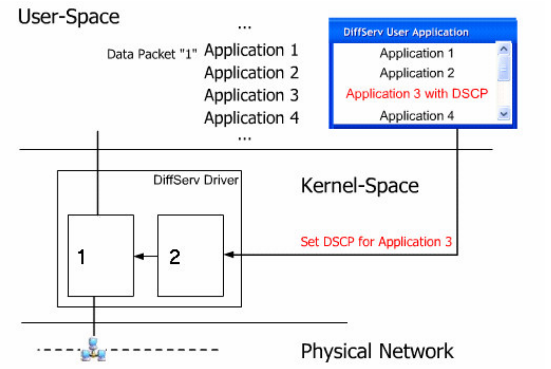


Fig.7 Interaction between the modules for the DiffServ Application

The last module is the Packet Capture, used to capture a selected number of frames for testing purposes. In the future it can be extended to an application for complete network monitor. The installation is simple, using two *.inf* files: one for the service (protocol) part of the Intermediate driver *(Class = NetService)* and the other for the miniport part of the driver *(Class = Net)*.

IV. EXPERIMENTAL RESULTS

Two categories of experiments were performed:

- experiments involving the use of the VoIP application
- experiments involving the use of the DiffServ Application

The VoIP application was investigated by using two different testbeds. The first one was a PC-to-PC configuration. One computer, having the IP address 192.168.1.1 is configured to run as a SIP server listening on the standard port 5060 and the other computer, having the IP address 192.168.1.2, connects as a client. A snapshot of the exchanged messages is presented in Fig.8a, Fig.8b and Fig.8c.

```
INVITE sip:server@192.168.1.1 SIP/2.0
Via: SIP/2.0/UDP 192.168.1.2:5060
From: client <sip:client@192.168.1.2>
To: server <sip:server@192.168.1.1>
Call-ID: 5978@192.168.1.2
CSeq: 1 INVITE
Content-Type: application/sdp
Content-Length: 90

v=0
o=client 5978 101 IN IP4 192.168.1.2
c=IN IP4 192.168.1.2
m=audio 35000 RTP/AVP 0
```

Fig.8a. SIP INVITE message sent by the client to the server

```
SIP/2.0 100 Trying
Via: SIP/2.0/UDP 192.168.1.2:5060
From: client <sip:client@192.168.1.2>
To: server <sip:server@192.168.1.1>
Call-ID: 5978@192.168.1.2
CSeq: 1 INVITE
Content-Length: 0
SIP/2.0 200 OK
Via: SIP/2.0/UDP 192.168.1.2:5060
From: client <sip:client@192.168.1.2>
To: server <sip:server@192.168.1.1>;tag=12345
Call-ID: 5978@192.168.1.2
Contact: <sip:server@192.168.1.1>
CSeq: 1 INVITE
Content-Type: application/sdp
Content-Length: 90

v=0
o=server 9876 101 IN IP4 192.168.1.1
c=IN IP4 192.168.1.1
m=audio 35000 RTP/AVP 0
```

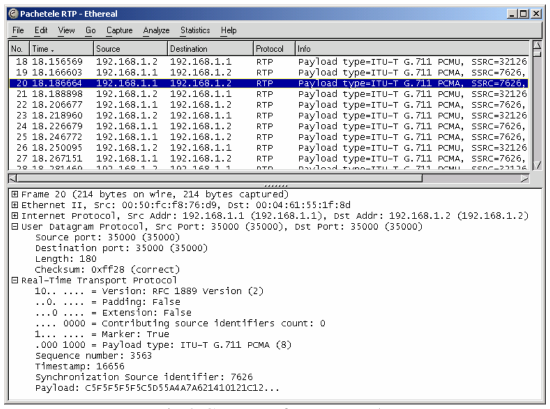
Fig.8b. SIP Trying and SIP OK messages sent by the server

```
ACK sip:server@192.168.1.1 SIP/2.0
Via: SIP/2.0/UDP 192.168.1.2:5060
From: client <sip:client@192.168.1.2>
To: server <sip:server@192.168.1.1>;tag=12345
Call-ID:  5978@192.168.1.2
CSeq: 1 ACK
```

Fig.8c. SIP ACK message sent by the client

The RTP flow of packets can start because the ports on both machines were identified: in this example, port 35000. The codecs were established too, in this case being G.711 μ law.

Fig. 9 presents a capture from Ethereal showing a RTP packet that transports 20 ms of speech (a payload of 160 bytes).

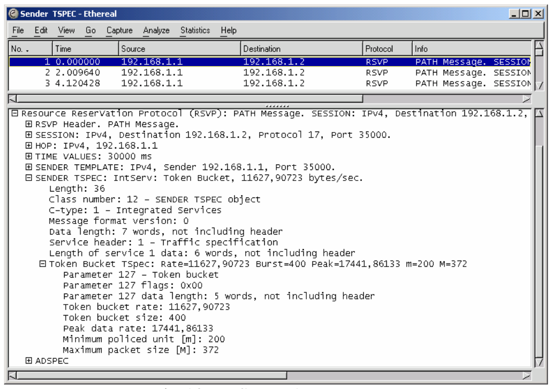


```
No. Time .     Source       Destination  Protocol Info
 18 18.156569  192.168.1.2  192.168.1.1  RTP      Payload type=ITU-T G.711 PCMU, SSRC=32126
 19 18.166603  192.168.1.1  192.168.1.2  RTP      Payload type=ITU-T G.711 PCMA, SSRC=7626,
 20 18.186664  192.168.1.1  192.168.1.2  RTP      Payload type=ITU-T G.711 PCMA, SSRC=7626,
 21 18.188898  192.168.1.2  192.168.1.1  RTP      Payload type=ITU-T G.711 PCMU, SSRC=32126
 22 18.206677  192.168.1.1  192.168.1.2  RTP      Payload type=ITU-T G.711 PCMA, SSRC=7626,
 23 18.218960  192.168.1.2  192.168.1.1  RTP      Payload type=ITU-T G.711 PCMU, SSRC=32126
 24 18.226679  192.168.1.1  192.168.1.2  RTP      Payload type=ITU-T G.711 PCMA, SSRC=7626,
 25 18.246772  192.168.1.1  192.168.1.2  RTP      Payload type=ITU-T G.711 PCMA, SSRC=7626,
 26 18.250095  192.168.1.2  192.168.1.1  RTP      Payload type=ITU-T G.711 PCMU, SSRC=32126
 27 18.267151  192.168.1.1  192.168.1.2  RTP      Payload type=ITU-T G.711 PCMA, SSRC=7626,
 28 18.281460  192.168.1.2  192.168.1.1  RTP      Payload type=ITU-T G.711 PCMU, SSRC=32126

⊞ Frame 20 (214 bytes on wire, 214 bytes captured)
⊞ Ethernet II, Src: 00:50:fc:f8:76:d9, Dst: 00:04:61:55:1f:8d
⊞ Internet Protocol, Src Addr: 192.168.1.1 (192.168.1.1), Dst Addr: 192.168.1.2 (192.168.1.2)
⊟ User Datagram Protocol, Src Port: 35000 (35000), Dst Port: 35000 (35000)
    Source port: 35000 (35000)
    Destination port: 35000 (35000)
    Length: 180
    Checksum: 0xff28 (correct)
⊟ Real-Time Transport Protocol
    10.. .... = Version: RFC 1889 Version (2)
    ..0. .... = Padding: False
    ...0 .... = Extension: False
    .... 0000 = Contributing source identifiers count: 0
    1... .... = Marker: True
    .000 1000 = Payload type: ITU-T G.711 PCMA (8)
    Sequence number: 3563
    Timestamp: 16656
    Synchronization Source identifier: 7626
    Payload: C5F5F5F5F5C5D55A4A7A621410121C12...
```

Fig.9 Capture of a RTP packet

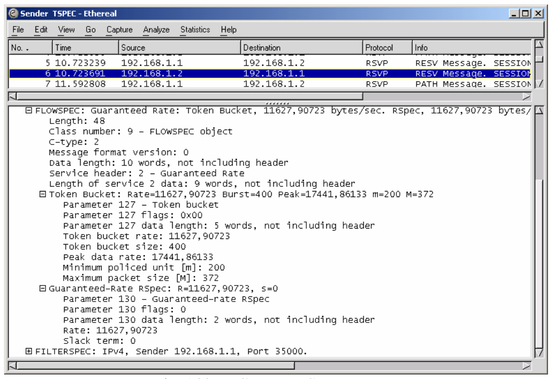
IntServ is used to perform QoS, by means of the Microsoft GQOS. The necessary parameters to be filled in the QoS structure [2] are computed taking into account that for about 20 milliseconds of speech, sampled at 8 kHz with 8 bits per sample, the RTP payload will be 160 bytes. Fig.10a presents a capture from a RSVP PATH message and Fig.10b presents a RSVP RESV message, both captured with Ethereal. Reservation takes place for the ports 35000 already established for RTP transfer (*TokenBucket=11627, Maximum packet size=372, Token Bucket size=400, Peak data rate=17441*).



```
No. .  Time      Source       Destination  Protocol Info
     1 0.000000  192.168.1.1  192.168.1.2  RSVP     PATH Message. SESSION
     2 2.009640  192.168.1.1  192.168.1.2  RSVP     PATH Message. SESSION
     3 4.120428  192.168.1.1  192.168.1.2  RSVP     PATH Message. SESSION

⊟ Resource ReserVation Protocol (RSVP): PATH Message. SESSION: IPv4, Destination 192.168.1.2,
  ⊞ RSVP Header. PATH Message.
  ⊞ SESSION: IPv4, Destination 192.168.1.2, Protocol 17, Port 35000.
  ⊞ HOP: IPv4, 192.168.1.1
  ⊞ TIME VALUES: 30000 ms
  ⊞ SENDER TEMPLATE: IPv4, Sender 192.168.1.1, Port 35000.
  ⊟ SENDER TSPEC: IntServ: Token Bucket, 11627,90723 bytes/sec.
      Length: 36
      Class number: 12 - SENDER TSPEC object
      C-type: 1 - Integrated Services
      Message format version: 0
      Data length: 7 words, not including header
      Service header: 1 - Traffic specification
      Length of service 1 data: 6 words, not including header
    ⊟ Token Bucket TSpec: Rate=11627,90723 Burst=400 Peak=17441,86133 m=200 M=372
        Parameter 127 - Token bucket
        Parameter 127 flags: 0x00
        Parameter 127 data length: 5 words, not including header
        Token bucket rate: 11627,90723
        Token bucket size: 400
        Peak data rate: 17441,86133
        Minimum policed unit [m]: 200
        Maximum packet size [M]: 372
  ⊞ ADSPEC
```

Fig.10a. RSVP PATH message



```
No. .  Time       Source       Destination  Protocol Info
     5 10.723239  192.168.1.1  192.168.1.2  RSVP     RESV Message. SESSION
     6 10.723691  192.168.1.2  192.168.1.1  RSVP     RESV Message. SESSION
     7 11.592808  192.168.1.1  192.168.1.2  RSVP     PATH Message. SESSION

  ⊟ FLOWSPEC: Guaranteed Rate: Token Bucket, 11627,90723 bytes/sec. RSpec, 11627,90723 bytes/
      Length: 48
      Class number: 9 - FLOWSPEC object
      C-type: 2
      Message format version: 0
      Data length: 10 words, not including header
      Service header: 2 - Guaranteed Rate
      Length of service 2 data: 9 words, not including header
    ⊟ Token Bucket: Rate=11627,90723 Burst=400 Peak=17441,86133 m=200 M=372
        Parameter 127 - Token bucket
        Parameter 127 flags: 0x00
        Parameter 127 data length: 5 words, not including header
        Token bucket rate: 11627,90723
        Token bucket size: 400
        Peak data rate: 17441,86133
        Minimum policed unit [m]: 200
        Maximum packet size [M]: 372
    ⊟ Guaranteed-Rate RSpec: R=11627,90723, s=0
        Parameter 130 - Guaranteed-rate RSpec
        Parameter 130 flags: 0
        Parameter 130 data length: 2 words, not including header
        Rate: 11627,90723
        Slack term: 0
  ⊞ FILTERSPEC: IPv4, Sender 192.168.1.1, Port 35000.
```

Fig.10b. RSVP RESV message

The second testbed for the VoIP application is presented in Fig.11 and involves a CISCO 1750 router, an analogue telephone and the proposed application running on a PC.

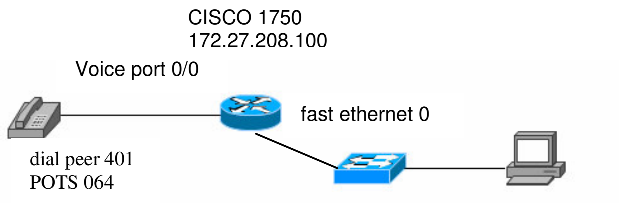
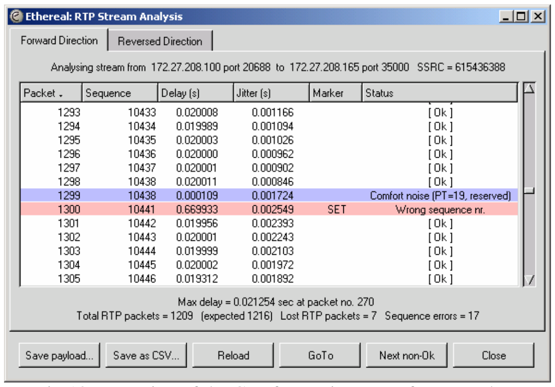


Fig.11. Testbed for the PC-to-Phone configuration

The developed application worked with minor modifications. For the RTP transmission, we used fixed-size packets. In the case of the RTP reception, we observed that the router detects the silence intervals and in order to preserve the bandwidth it sends a RTP “Comfort Noise” (Payload Type=19). Version 0.10.3 of the Ethereal application correctly detects this type of packet (selecting Statistics/ RTP/ Stream Analysis) as presented in Fig.12. There are two solutions to this situation. The first solution is to disable VAD (Voice Activity Detection) at the router side using the following command:

```
R1750(config-dial-peer)# no vad
```

The other solution is to detect within the VoIP application the "Comfort Noise" RTP packet and stop playing the buffer until a new valid RTP packet appears.



Analysing stream from 172.27.208.100 port 20688 to 172.27.208.165 port 35000 SSRC = 615436388

Packet	Sequence	Delay (s)	Jitter (s)	Marker	Status
1293	10433	0.020008	0.001166		[Ok]
1294	10434	0.019989	0.001094		[Ok]
1295	10435	0.020003	0.001026		[Ok]
1296	10436	0.020000	0.000962		[Ok]
1297	10437	0.020001	0.000902		[Ok]
1298	10438	0.020011	0.000846		[Ok]
1299	10438	0.000109	0.001724		Comfort noise (PT=19, reserved)
1300	10441	0.669933	0.002549	SET	Wrong sequence nr.
1301	10442	0.019956	0.002393		[Ok]
1302	10443	0.020001	0.002243		[Ok]
1303	10444	0.019999	0.002103		[Ok]
1304	10445	0.020002	0.001972		[Ok]
1305	10446	0.019312	0.001892		[Ok]

Max delay = 0.021254 sec at packet no. 270
Total RTP packets = 1209 (expected 1216) Lost RTP packets = 7 Sequence errors = 17

Fig.12. Detection of the Comfort Noise type of RTP packets

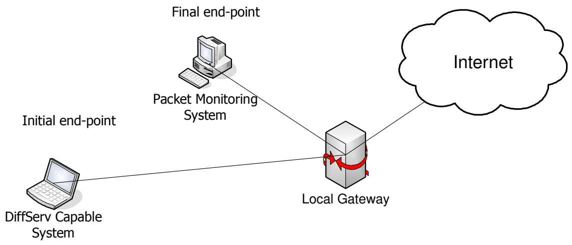
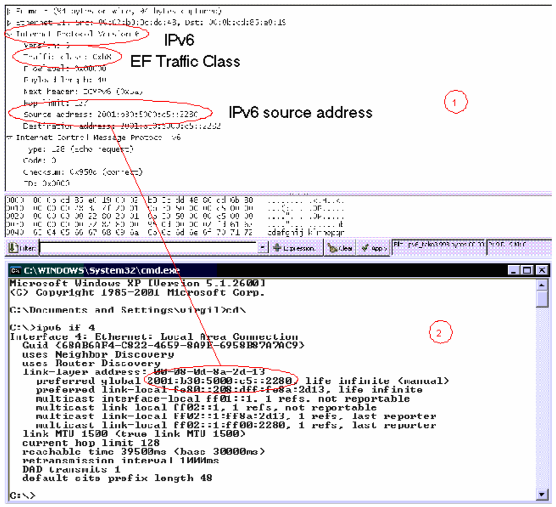


Fig.13. The testbed used for the DiffServ application

The second part of the experiments is dedicated to DiffServ, the testbed being described in Fig.13. One experiment performed in the case of IPv6 traffic was to mark ICMPv6 traffic as EF. Fig.14/1 represents an Ethereal-based capture and Fig.14/2 represents output from *ipv6 if* command showing the local IPv6 address on the DiffServ capable system.



```
▷ Frame [illegible] (94 bytes on wire, 94 bytes captured)
▷ Ethernet II, Src: 00:02:b3:3c:dc:43, Dst: 00:0b:cd:85:a0:15
▽ Internet Protocol Version 6
    Version: 6
    Traffic class: 0xb8
    Flowlabel: 0x00000
    Payload length: 40
    Next header: ICMPv6 (0x3a)
    Hop limit: 127
    Source address: 2001:b30:5000:c5::2280
    Destination address: 2001:b30:5000:c5::2282
▽ Internet Control Message Protocol v6
    Type: 128 (Echo request)
    Code: 0
    Checksum: 0x950c (correct)
    ID: 0x0000

0000  [illegible]
0010  [illegible]
0020  [illegible]
0030  [illegible]
0040  63 64 65 66 67 68 69 6a  6b 6c 6d 6e 6f 70 71 72   cdefghij klmnopqr
```

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

C:\Documents and Settings\virgil>cd\

C:\>ipv6 if 4
Interface 4: Ethernet: Local Area Connection
  Guid {68AB6AF4-C822-4659-8A9E-6958B87A7AC9}
  uses Neighbor Discovery
  uses Router Discovery
  link-layer address: 00-08-0d-8a-2d-13
    preferred global 2001:b30:5000:c5::2280, life infinite (manual)
    preferred link-local fe80::208:dff:fe8a:2d13, life infinite
    multicast interface-local ff01::1, 1 refs, not reportable
    multicast link-local ff02::1, 1 refs, not reportable
    multicast link-local ff02::1:ff8a:2d13, 1 refs, last reporter
    multicast link-local ff02::1:ff00:2280, 1 refs, last reporter
  link MTU 1500 (true link MTU 1500)
  current hop limit 128
  reachable time 39500ms (base 30000ms)
  retransmission interval 1000ms
  DAD transmits 1
  default site prefix length 48

C:\>
```

Fig.14. Marking ICMPv6 traffic as EF

Other DiffServ experiments were performed with applications running over IPv4.

V. CONCLUSIONS AND FUTURE WORK

In this paper we described two applications. The first one was a VoIP application using SIP and RSVP. The second one was a DiffServ program that enables QoS even for QoS-unaware applications. We tested the interoperability of these applications with existing implementations, mainly with those existent within the Cisco routers. For future work we intend to add security to the VoIP application and to add IEEE 802.1p/Q to the NDIS Driver.

REFERENCES

[1] J. Rosenberg et al, "SIP: Session Initiation Protocol", *RFC3261*, June 2002.

[2] J. Handley, V. Jacobson, "SDP: Session Description Protocol", *RFC 2327*, April 1998.

[3] D. Zinca, V. Dobrota, C.M. Vancea, G. Lazar – A Practical Evaluation of QoS for Voice over IP. *12th IEEE Workshop on Local and Metropolitan Area Networks, LANMAN 2002*, 11-14 August 2002, Stockholm-Kista, Sweden, pp. 65-69.

[4] D. Zinca, V. Dobrota, C.M. Vancea, G. Lazar, "Developing QoS-aware Applications in LANs", *Buletinul Universitatii "Politehnica", Seria Electrotehnica, Electronica si Telecomunicatii*, Tom 47 (61), 2002, Fascicola 1-2, 2002, pp. 150-153.

[5] J. Wroclawski, "The Use of RSVP with IETF Integrated Services" , *RFC 2210*, September 1997.

[6] S. Shenker et al, "Specification of Guaranteed Quality of Service", *RFC 2212*, September 1997.

[7] H. Schulzrinne et al, "RTP: A Transport Protocol for Real-Time Applications", *RFC 1889*, January 1996.

[8] A. Takahashi, H. Yoshino, "Perceptual QoS Assessment Technologies for VoIP", *IEEE Communications Magazine*, July 2004, pp. 28-34.

[9] K. Nichols, V. Jacobson, L. Zhang, "A Two-bit Differentiated Services Architecture for the Internet", *RFC 2638*, July 1999.

[10] A. Charny et al, Supplemental information for the New Definition of the EF PHB (Expedited Forwarding Per-Hop behaviour), *RFC3247*, March 2002.

[11] S. Blake et al, "An Architecture for Differentiated Services", *RFC 2475*, December 1998.

[12] D. Grossman, "New Terminology and Clarifications for DiffServ", *RFC 3260*, April 2002.

[13] K. Nichols et al, "Definition of the Differentiated Service Field (DS Field) in the IPv4 and IPv6 headers", *RFC 2474*, December 1998.

[14] ***, Microsoft Developer Network Library for Visual Studio .NET 2003

[15] http://www.ndis.com

[16] http://www.ethereal.com