INTERNATIONAL JOURNAL OF COMMUNICATION SYSTEMS
Int. J. Commun. Syst. 2005; **18**:847–866
Published online 6 April 2005 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/dac.735

Porting and performance aspects from IPv4 to IPv6: The case of OpenH323

Ch. Bouras[1,2,*,†], A. Gkamas[1,2,‡], D. Primpas[1,2,§] and K. Stamos[1,2,¶]

[1] *Research Academic Computer Technology Institute, Riga Feraiou 61, GR-26221 Patras, Greece*
[2] *Computer Engineering and Informatics Department, University of Patras, GR-26500 Patras, Greece*

SUMMARY

This paper is a summary of our experiences on a case study for porting applications to IPv6. We present the results of the effort to port OpenH323, an open-source H.323 platform to IPv6, which we believe can serve as guidelines for other projects with similar goals. We briefly present the structure of the OpenH323 platform. We also discuss a number of issues arising during the porting of a platform to IPv6, like which would be the easiest approach to the porting procedure, how compatibility with earlier, IPv4-only versions of the platform could be retained, if there are any useful tools for assisting this task, how and when one could be positive that the necessary modifications had been made, and which testing procedures should be followed. We then present a variety of experiments that we conducted in order to comparatively evaluate the IPv4 and IPv6 protocol stacks. We also present the results of some initial experiments comparing IPv4 and IPv6 performance under congested network links and the conclusions that they lead us to. Copyright © 2005 John Wiley & Sons, Ltd.

KEY WORDS: IPv6; H.323; real-time applications; videoconferencing; QoS

1. INTRODUCTION

The new version of IP, IPv6 [1], constitutes an effort to overcome the inborn limitations of IPv4, so that the new protocol is able to respond to the new needs of today's Internet.

In addition to the upgrade to 128-bit addresses, the IPv6 packet format was redesigned in order to overcome the limitations of IPv4. More than simply increasing the address space, IPv6 offers the following improvements:

- IPv6 has built in security support.

*Correspondence to: Christos Bouras, Research Academic Computer Technology Institute, Riga Feraiou 61, GR-26221 Patras, Greece.
† E-mail: bouras@cti.gr; URL: <http://ru6.cti.gr/bouras>
‡ E-mail: gkamas@cti.gr
§ E-mail: primpas@cti.gr
¶ E-mail: stamos@cti.gr

Contract/grant sponsor: European Commission, 6NET IST project; contract/grant number: IST-2001-32603

Received 1 June 2004
Revised 1 February 2005
Accepted 1 February 2005

- IPv6 eliminates the checksum from the IP header.
- IPv6 is more flexible and extensible than IPv4.
- IPv6 facilitates efficient renumbering of sites by explicitly supporting multiple addresses on an interface.
- IPv6 supports plug and play operation.

The transition phase from IPv4 to IPv6 has raised many discussions among the Internet community, as a lot of companies and network administrators are reluctant, facing what they perceive as a great challenge with large costs. Apart from the network and hardware part of the issue, a very important aspect is the modification (porting) of existing applications so that they become IPv6 enabled [2]. It is a necessary step in the wider adoption of IPv6, not only because without them the new infrastructure becomes useless for the user, but also because applications have the ability to clearly demonstrate the advantages of IPv6. The majority of network applications in existence today presume the use of the IPv4 protocol, so the transition to IPv6 has to be accompanied by the development of new applications and/or the modification of the existing ones, so that they can be used in IPv6 environments. It has often been demonstrated that the difficulty of modifying existing applications varies significantly from one case to another. Depending on the specific application at hand, it can be either a relatively quick and easy process [3] or a more complicated and resource-consuming task [4]. However, there are modifications that are needed to port applications to IPv6, which are likely to occur in most of the cases. In this paper we report our experiences from porting to IPv6 the library upon which the OpenH323 project is based, a large open-source library.

We believe that the OpenH323 project is a very useful project and suitable application for IPv6 due to a number of reasons:

- Teleconferencing applications are going to play a large role in the future high-speed networks, and it is therefore interesting to experiment with the impact of IPv6, the future Internet Protocol, on these applications.
- The OpenH323 project is based on a large library (the OpenH323 library) that offers a high-level interface to applications. When this library is made compatible with IPv6, it can be used for quickly building additional applications.
- It presents a problem that can be approached by different angles, therefore allowing us to compare different strategies on the porting issue.
- It is written in C++, which is the most widely used language for teleconferencing stacks and applications, and network applications in general.

Part of this work has been presented in References [5–7].

The rest of this paper is organized as follows: Section 2 discusses related work in the interest area of this paper. Section 3 gives a brief overview of the OpenH323 project structure. Section 4 presents the different ways of approaching the porting task, and the various ways of maintaining backwards compatibility with IPv4. Section 5 summarizes the procedure for our porting efforts and Section 6 presents in detail the most significant modifications that had to be made to the source code. Section 7 discusses some of the problems we faced. Section 8 addresses the problem of verifying when porting is correctly completed. Section 9 then discusses the criteria with which an IPv6-enabled application could be evaluated. These criteria are used to draw conclusions from the experiments that are described in Section 10. These conclusions are presented in Section 11, and Section 12 gives an overview of the directions towards which our future work is going to be headed.

2. RELATED WORK

The problem of porting existing applications to IPv6 has been so far addressed by several researchers, including companies and academic institutes. A white paper by Microsoft [8] focuses on Windows applications, but at the same time offers some general guidelines that apply to any application for any operating system. In Reference [9], the authors emphasize more on some general knowledge that a programmer must acquire before dealing with the problem of porting applications to IPv6, than on presenting step-by-step instructions. There are also books [10] and online sources [11] that can provide useful assistance to a programmer on this task. Recently, a paper by Robles *et al*. presented the authors' results from porting a SIP implementation to IPv6 [12]. We can make useful comparisons with their work, since it deals with SIP, a competing protocol to H.323, and since they worked with Java, an alternative language to C++ for developing various kinds of applications. Also Reference [13] deals with issues that a programmer faces when the porting of an application to the IPv6 protocol is undertaken. In Reference [6] the authors report a very positive experience from porting the well-known Quake game to IPv6. Recently, SAP have announced their work on porting the SAP Web Application Server [7], which is also a large undertaking, considering the scale of the software and its widespread dependence on networking components. Furthermore, a number of research projects (6NET [14], Euro6IX [15], 6INIT [16], KAME [17]) are actively investigating the migration effort and the benefits from IPv6, and have shared or are going to share their valuable experiences. In the framework of the 6NET project, Reference [18] offers valuable experience regarding the IPv6-enabled version of the Globus toolkit. CTI is one of the participants of the 6NET project, and this work was partially supported by the 6NET project [14].

Libpnet6 [19] is another interesting open-source project which aims at developing 'a powerful library for writing cross-platform network applications'. It is a library that is written with IP-level transparency from the very beginning, and is also portable across a wide range of operating systems. It is written in ANCI-C and offers extensive functionality for network applications.

3. STRUCTURE OF THE OpenH323 LIBRARY

The OpenH323 project [20] develops a central library, the OpenH323 library, with the purpose of creating 'a full featured, interoperable, open-source implementation of the ITU H.323 teleconferencing protocol that can be used by personal developers and commercial users without charge'. The OpenH323 project started in September 1998 by Equivalence Pty Ltd., a private company based in Australia, and its code is distributed under the MPL (Mozilla Public license). The open-source OpenH323 library can be used for the rapid development of applications that wish to use the H.323 protocol [21] for multimedia communications over packet-based networks. It is written with C++, and currently contains nearly 100 classes in over 350.000 lines of source code. There are classes that represent an H.323 connection, various types of H.323 channels, gatekeeper and transport protocols. The main classes in the OpenH323 library are

- H323EndPoint: an application based on the OpenH323 library typically has one instance of a descendant of this class.
- H323Listener: this class represents a listener thread on a transport protocol that monitors its protocol waiting for incoming calls.

- H323Transport: descendants of this class represent particular transport protocols like H323TransportTCP and H323TransportUDP. An instance of a descendant of this class is created upon detection of a new call.
- H323Connection: represents a connection between two endpoints that has been established after a successful call.
- H323Negotiator: this class has a number of descendant classes that are used to maintain the state and functionality of each command or variable defined by the H.245 protocol.
- H323Channel: represents a logical channel used to carry data between two endpoints.

Internally, the OpenH323 classes do not directly make use of system libraries. Instead, when they want to use an operating system mechanism (e.g. sockets, threads, GUI, I/O), they make calls to another open-source library called PWLib. PWLib has also been developed by Equivalence Pty Ltd. and is licensed under the MPL. It contains classes that encapsulate I/O, GUI, multi-threading and networking functionality, and also classes that represent basic 'container' classes such as arrays, linear lists, sorted lists (RB Tree) and dictionaries (hash tables). Being such a general-purpose library results in a source code base of over 300 classes and almost 150.000 source code lines. The goal of the PWLib library is, by providing the necessary operating system abstractions, to support applications that can run both on Microsoft Windows

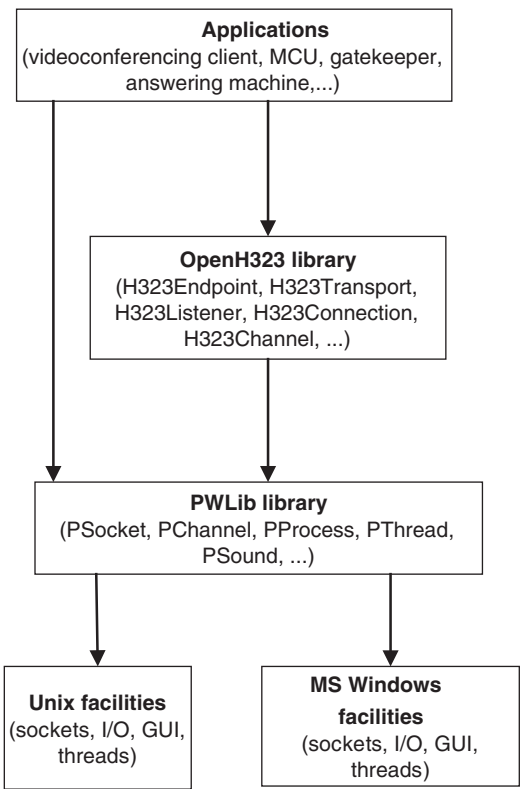


Figure 1. Relationship of the OpenH323 and PWLib libraries.

and Unix systems, without modifying the source code. By being based on the PWLib, the OpenH323 library manages to be portable between Windows and Unix systems. For our porting purposes, this meant that we had to examine both libraries (OpenH323 and PWLib) for IPv4 dependencies.

A number of applications have been developed on top of the OpenH323 library, both within and outside the OpenH323 project. They include a command line H.323 client, an H.323 videoconferencing server (MCU), H.323 answering machine, H.323 gatekeeper, H.323 to PSTN and fax modem to T.38 gateways, and GnomeMeeting [22], a graphical H.323 client for Linux.

Figure 1 gives a visual representation of the way the OpenH323 and PWLib libraries interconnect and the architecture of the applications developed on top of these two libraries.

4. BACKWARDS COMPATIBILITY WITH IPv4

Whenever an application has to be made IPv6 aware, there are a number of choices over the way that this can be achieved. It is important to know whether there is going to be a need for backwards compatibility with IPv4. For the vast majority of applications, this is the case, since there is going to be a long period of transition from IPv4 to IPv6, during which IPv6-enabled nodes will have to communicate with IPv4 nodes, and will probably implement a dual stack. The alternatives for the porting task can be categorized as follows:

- Two source code bases, two binaries (IPv4-dependent and IPv6-dependent binaries).
- Single source code base, two binaries (IPv4-dependent and IPv6-dependent binaries).
- Single source code base, single binary (IP version agnostic binary).

The simplest approach is to create a totally new version of the application by changing the IPv4-dependent parts of the code with their IPv6 counterparts, thus making the code IPv6-enabled, but again IP protocol dependent. This way, two independent applications will have to be maintained, one for use with IPv4-only nodes and another for use with IPv6 nodes. Another similar approach would be to incorporate the necessary changes in the same source file with the original IPv4-only version using preprocessor directives that will build either an IPv6-compatible binary version, or the original IPv4-only version. The crucial benefit over the previous approach is that there is only one code base to maintain. The third approach is to substitute the IP protocol-dependent parts with IP Version-Agnostic source code, thus making the application capable of handling any type of IP protocol (IPv4 and IPv6). The last approach is obviously much more appealing because it eliminates the complexities of having two source code bases or two different binaries with probably the same name. The downside is that it requires the most extensive modifications in the source code and probably in the program's logic. Our opinion is that for large projects a more gradual approach might be more appropriate, always depending on the needs that drive a porting effort. We therefore began by porting the project to IPv6-dependent code. The benefit is that such an initial approach is easier to carry out, requires less modification and intervention in already-tested source code and reveals all the dependencies of the source code. With the experience gained, the porting can then be extended in order to make the application in question IP protocol independent.

It is interesting to consider whether an application running on a dual-stack host can communicate with an earlier IPv4-only version of the application also running on a dual-stack host. By using the mechanism of IPv4-mapped IPv6 addresses an IPv6-enabled application

Table I. Interoperability between IPv4 and IPv6 versions running on dual-stack hosts.

	IPv4 server	IPv6 server
IPv4 client	Communicate using IPv4	Communicate using IPv4, server sees IPv4-mapped IPv6 address
IPv6 client	Can communicate if the IPv6 client uses an IPv4-mapped IPv6 address	Communicate using IPv6

operating as a server can communicate with an IPv4-only version operating as client. An IPv6 client can communicate with an IPv4 server only if it uses its IPv4-mapped IPv6 address. This can be achieved by using the DNS (Domain Name Service) mechanism and choosing the A record, which is returned to the client as the server's IPv4-mapped IPv6 address. These observations are summarized in Table I.

Communication between IPv4-only and IPv6 nodes can be achieved using proxies and other application layer gateways that are going to be used in the transition phase from IPv4 to IPv6.

5. METHODOLOGY

Below we present the procedure that can be used during efforts to port applications to IPv6. It is intended as a guideline for projects that deal with porting a similarly large library to IPv6:

- Parse the source code with an automatic tool like Checkv4.exe [23].
- Modify the source code lines reported by the automatic tool. Most of these changes have to do with replacing the data structures used for storing addresses, replacing network functions for DNS access, address transformation, and replacing the constants that have been changed for IPv6.
- Make any other necessary modifications in more subtle places not reported by the automatic tool.
- Test and debug the code, correcting any issues that arise.
- Verify completeness of porting effort.

The first step is to thoroughly read and understand the source code in order to become familiar with the overall structure and techniques used, especially if the person who undertakes the porting task has not been involved in the initial development of the project. The special characteristics of the specific project will determine the most appropriate approach to the porting task. For the initial phases of the porting an automatic tool that parses the source code and reports the source code lines that contain IPv4-dependent code can prove very useful. The relevant changes are probably rather straightforward, and can proceed in a mechanistic way. There are a number of tools of this kind available, most of which are free and cover a wide range of platforms, like Microsoft's Checkv4 [23] for MS Windows, Sun's Socket Scrubber [24] for Solaris and Compaq's IPv6 Porting Assistant [25] for Tru64 Unix. They are, however, suitable for code written in the C\C++ languages; for other languages there is still a lack of such tools. While C\C++ are dominant in the area of low-level network programming, there is definitely need for similar tools for less mainstream languages. What these tools mainly do is search for

patterns that are generally recognized as potential points that need modification. So in many cases there is still work to be done by the programmer after the lines suggested by the automatic tool have been modified. This phase largely depends on the extent and the structure of the networking code, and the degree to which IPv4-dependent logic is scattered within the source code. It is very likely that some indirect IPv4 dependencies will be discovered through trial and error, and revisiting parts of the code that were not immediately obvious that had to be modified, will be necessary. The last step is the verification of the porting effort. The programmer has to determine a number of test programs that make use of all the affected functionality and verify their correct operation, before porting can be considered completed.

6. NECESSARY MODIFICATIONS

In order to port the OpenH323 project we followed the steps described in Section 5. Since we had to port source code that we were not familiar with, the first step in our effort was to become familiarized with the structure of the OpenH323 and PWLib libraries and locate the classes that obviously had to be modified. In our case, such were the PWLib classes that encapsulated the socket mechanism functionality (thus making the applications that used the PWLib, such as the OpenH323 library, portable). These classes (inheriting from a class named PSocket) had to be redesigned since they dealt with the socket mechanism and contained low-level details. This design pattern by the PWLib authors guaranteed to some extent, however, that the IP dependencies would be quite limited to these classes that directly manipulated IP-dependent socket structures and function calls. All other classes in the OpenH323 and PWLib libraries make use of the facilities offered by the PSocket class and its descendants.

There are a number of quite straightforward modifications that had to be carried out during the porting of the OpenH323 project to IPv6. The most important are:

- Changing data structures that encapsulate IP addresses, and that have to be sufficiently enlarged in order to cope with 128-bit IPv6 addresses. So the sockaddr_in structure was replaced with the 32-byte sockaddr_in6 structure.
- Replacing IPv4 constants like INADDR_ANY and AF_INET with their IPv6 counterparts.
- Replacing function calls that are IPv4 specific with their IPv6-capable counterparts. Functions like inet_ntoa, inet_aton, inet_addr, gethostbyname had to be replaced.
- Replacing hard-coded IPv4 addresses like the loopback address with IPv6 addresses, or eliminate them altogether by properly modifying the source code.
- Replace any IPv4-only options with their IPv6 counterparts or delete the corresponding functionality altogether. Although not used in practice, we had to modify code for setting and retrieving the TOS field in IPv4.

Because of the large size of the code base of the OpenH323 and PWLib libraries we used an automated tool, in order to trace down the most obvious IP protocol-dependent points in the source code. The tool we chose was Checkv4.exe by Microsoft [23], which is offered as part of the experimental IPv6 stack for Windows 2000.

After these two phases of the porting effort had been successfully completed, we started testing the code base in an IPv6 environment, in order to verify the correctness and

completeness of our porting work. The length and difficulty of this phase largely depends on the special characteristics of the application that has to be ported, and the provision taken (if any) to eliminate indirect IPv4 dependencies or to structure the code in such a way that they can be easily traced and modified.

7. PROBLEMS

There are two kinds of issues that introduce difficulties for the porting effort: isolating the classes and functions that have to be modified, and the fact that some of the indirect dependencies might be scattered or affect large portions of the source code. In this section we report some of the most difficult issues that were revealed during the porting of the OpenH323 project that required changes not easily identifiable from the beginning.

- There were many parts in the code where the IP address was indirectly assumed to have a 4-byte length. This included the size of arrays, the number of repetitions for loops and the variables used. Most of these changes were concentrated in classes like PSocket and its descendant PIPSocket in the PWLib library that directly dealt with IP addresses and contained numerous functions in order to manipulate them in various ways. There were also a few dependencies of this kind scattered in a lot of other classes, mainly in the PWLib library. Because such dependencies are not detectable using an automated tool and because of the very large size of the source code base, this was the most time-consuming part of the modifications in the source code and also the cause for a number of bugs that appeared during the porting.
- The socket API implementations of Linux and Windows, although both based on the original Berkeley Software Distribution (BSD) sockets paradigm are not fully compatible [26]. Furthermore, the Windows IPv6 stack we used, which was included in the Microsoft IPv6 Technology Preview for Windows 2000 did not support some of the socket interface extensions for IPv6, as proposed in RFC 3493 [27]. For example, the functions getipnodebyname and getipnodebyaddr (used for node name to address translation and vice versa) are not supported, and neither are the inet_ntop and inet_pton functions that perform address conversions between binary and text form. The Windows IPv6 implementation also does not support IPv4-mapped addresses as described in RFC 3493 [27], and has a small number of other differences, that required special handling in order to maintain the compatibility of the platform with both operating systems (Linux and Windows).
- A common phenomenon in network applications is the difference between various computer architectures in the order they store 2-byte values. Linux uses little endian (high-order byte at higher address) while the Internet Protocols define big endian for network byte order. This difference caused some implications in the way IPv6 addresses were stored and manipulated that did not appear with the 4-byte IPv4 addresses, since IPv6 addresses comprise of eight 2-byte fields.
- The size of the source code base (around half a million lines for both PWLib and OpenH323) is an important factor, although the well-structured design of the library alleviated its effect. In the following paragraph, we access the problem of determining when all points that needed porting had been successfully modified.

8. VERIFYING PORTING COMPLETION

A question that quickly arose during our efforts to port such a large project as OpenH323 to the IPv6 protocol was how to verify that our work had been completed successfully and correctly. In smaller, single-purpose applications this is probably not an issue, because the range of functionality that has to be tested is relatively small. The OpenH323 library, however, is a large library that can support a number of independent applications. Moreover, since the OpenH323 library makes use of the facilities offered by the even larger PWLib library, this library also had to be included in our porting efforts. We had, therefore, to inspect and often modify a large number of classes and functions. We also extended the porting to include a wide range of applications that were based on the OpenH323 library. Because most of these applications use the advanced functionalities offered by the central library, they only have to deal with high-level issues and the IP dependencies are hidden for the applications inside the library. This means that most of the modifications in the applications' source code were relatively small and only had to do with hard-coded IPv4 addresses.

In general, there are a number of testing strategies that we followed [28]:

- High-level testing: this testing strategy is initially targeted towards the high-level view of a system. It emphasizes on testing applications that use a wide range of functionality from the supporting libraries, and can therefore reveal the way different parts of the system interoperate. This method is very useful for acquiring a larger, more general picture of the system.
- Low-level testing: the opposite approach is to try and isolate specific classes and methods and try to test their behaviour by using simple test applications with limited functionality. This way, errors can be more easily identified and their origin can be more easily attributed.
- Comparative (back-to-back) testing: this strategy can be used when different versions of the same system are available (as was our case, with an IPv4-only version, and an IPv6-enabled version). The two versions can be tested together and their operation can be compared.

For our purposes, we used a combination of the three techniques outlined above, with emphasis on the third approach (back-to-back testing). The fact that our goal was to modify an already functioning system meant that back-to-back testing was very important, both in determining whether an application operated as should be expected, and in tracing down the point in execution where an error appeared.

In order to verify the completeness of the porting, we had to define a set of applications that would make use of all the affected functionality inside the libraries. The already developed applications were initially used, because they cover a very wide spectrum of the OpenH323 library functionality. They also avoid low-level details (and therefore further modifications) by using the high-level abstractions offered by the OpenH323 library. We compared the operation of these applications using the original IPv4-only libraries with the same applications (with any necessary modifications) using the IPv6-enabled libraries. In order to make the comparison we recorded debugging information from the modified and unmodified versions into files.

The most important part was testing the PWLib library, because that is where the main networking classes are located. Although many parts of its functionality were left totally

unchanged, the wide range of functions offered by the heavily modified classes like PIPSocket meant that we had to test those classes in many varying contexts. High-level testing was unable in this case to include every aspect of the affected functionality, so we were primarily based on comparative testing and low-level testing using simple proof-of-concept applications under various circumstances, trying to make sure that all possibilities are covered in accordance with the latest IPv6 specifications. For example, IPv6 addresses have to comply with the standard, 8-part format separated with colons (for example 3ffe:2c00:1b4a:af12:feb3:12:abab:20), but also with the shortened notation (2001:fffe::14ab) and the IPv4-compatible IPv6 addresses (x:x:x:x:x:x.150.140.141.17).

9. PERFORMANCE CRITERIA

In order to perform comparative testing, we first had to identify the criteria for evaluating the IPv4 and IPv6 versions of the application under test. Mainly due to the larger IP header, IPv6 can be expected to introduce some overhead compared to IPv4. Comparing the overhead caused by IPv6 vs the overhead by IPv4 is a difficult task, because a lot of factors are involved. Sometimes overhead can be attributed to a less-than-optimal implementation of the specific application with regard to IPv6. Another factor is the TCP/IP stack itself and the way it has been implemented. The DNS resolver can also play a small role, usually against IPv6 because of the additional AAAA record. It is also clear that when considering tunneling transition mechanisms, they will contribute to degraded performance for IPv6, since IPv6 packets have to be encapsulated in IPv4 packets and suffer the additional overhead. Perhaps the most important criterion is the final user perception that the application will give. Although it is highly subjective and can be influenced from a lot of factors (many of which are outside of the control of the application or the IPv6 stack implementation), it is important because it is connected with the acceptance of the IPv6 protocol. The main characteristic that determines the user perception when considering an IPv6 application and its IPv4 counterpart is usually the achieved throughput by each application version. For judging the quality of teleconferencing application, we can also use objective quality measurement methods, such as PESQ (perceptual evaluation of speech quality [29]) for voice or JNDmetrix [30] for video. We are also interested in the system administrator's perception, with regard to the ease of managing an IPv6-enabled application. This parameter is influenced a lot by the path taken for the porting: the development of a new application executable, or the simultaneous support of both IP versions by the same executable.

In order to evaluate the IPv6-enabled version of the OpenH323 project, we conducted a number of tests. The tests that follow took place on a real IPv6 testbed network. This testbed has been created internally in CTI and is displayed in Figure 2. Tests were carried out so that communication from one endpoint to the other had to pass through 2 hops, with the bottleneck link being the 10 Mbps one.

In order to create background traffic, we used the Iperf tool [31], which is capable of producing TCP/UDP traffic in both IPv4 and IPv6. For retrieving and studying the transmission/reception data we used both the RTP/RTCP feedback from the OpenH323 library, and the Sniff'Em network monitoring tool [32].

Transmission of video data was made using the OpenH323 built-in H.261 codec with CIF resolution, which, although optimized for low data rates and low motion and therefore

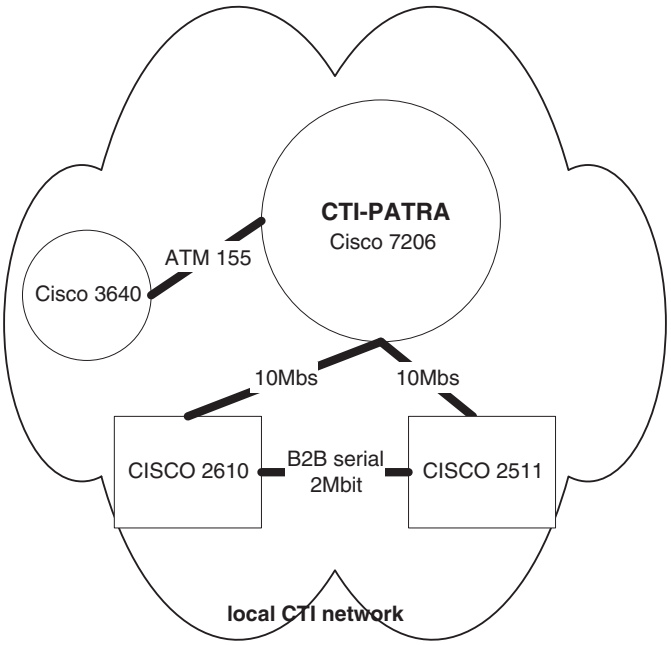


Figure 2. Internal testbed for running the tests.

producing lower quality results than H.263, was sufficient for our purposes. Audio transmission was achieved using G.711 (muLaw variation), a PCM scheme that operates at the rate of 64 Kbps.

The applications used for the tests were the OpenPhone GUI client, the OpenMCU implementation of a software MCU, and the OpenWAV application for transmitting pre-recorded audio files.

10. EXPERIMENTS AND ANALYSIS

10.1. Experiment 1: IPv4 and IPv6 communication with no competing traffic

Our first experiment was to test the IPv4 version of the OpenPhone application at a point-to-point communication, sending video and audio between 2 PCs, and without any competing traffic at the intermediate link. This experiment was designed in order to test the basic operation of the OpenH323 protocol stack on a non-congested network using the IPv4 protocol, and to have a reference point for the rest of the experiments we subsequently conducted.

As shown in Figure 3, we obtained a steady transmit rate of 16 KBytes/s throughout the experiment. The quality of the video transmitted was relatively low, because of the characteristics of the H.261 codec.

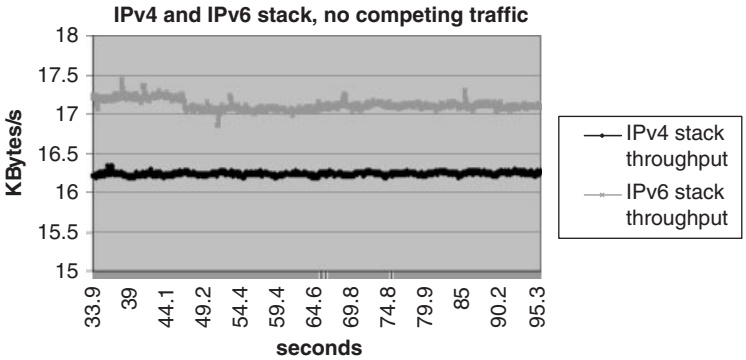


Figure 3. OpenPhone operation without competing traffic.

We then repeated the experiment using the IPv6 stack for the point-to-point communication between the two endpoints. Again, we were sending video and audio between 2 PCs, and without any competing traffic at the intermediate link. This experiment was also designed in order to test the basic operation of the OpenH323 protocol stack on a non-congested network using IPv6.

Again we can see in Figure 3 that in the absence of any competing traffic and with a link of much higher capacity than the H.261 codec could ever want, we obtain a steady transmission rate of around 17 KBytes/s, around 7% larger than the IPv4 transmission rate. This difference is due to the fact that the Data-Link layer was carrying 294-byte packets in the case of IPv4, and 314-byte packets in the case of IPv6. The standard IPv6 header is 20 bytes larger than the standard IPv4 header, which produces the 7% overhead. This is in fact an expected and known result, since the larger IPv6 header introduces some overhead, especially in relatively low-rate transmissions.

In both cases we can observe that the choice of network layer stack is not an issue, since the application will consume the required bandwidth, given an uncongested link. We cannot, however, expect that this will always be the case. A transmission rate of around 140 Kbits/s means that for low bandwidth links (for example modem or basic ISDN links) there will be significant congestion. Also for high bandwidth links that carry a lot of additional traffic, unwanted results can occur if the H.323 traffic is added to the competition. In the following experiments we experimented with the latter case, and we also tried to identify possible behaviour differences between IPv4 and IPv6.

In order to make sure that the choice of codec does not affect the results, we repeated the above experiments using only audio transmission from a simple H.323 VoIP application with the G.711 A-Law codec. The speech signal transmitted using the H.323 client was a short (20 seconds duration) phrase read from the same person in both (IPv4 and IPv6) tests. As shown in Figure 4, where the transmission rate is measured in intervals of 0.05 s, IPv6 maintains a data rate at around 50 Kbps (6.2 KBytes/s), while IPv4 maintains a data rate at around 47 Kbps (5.8 KBytes/s), almost 7% lower, the same result as with the H.261 experiment in Figure 3. The transmission rate for both protocols is below the rate needed by the G.711

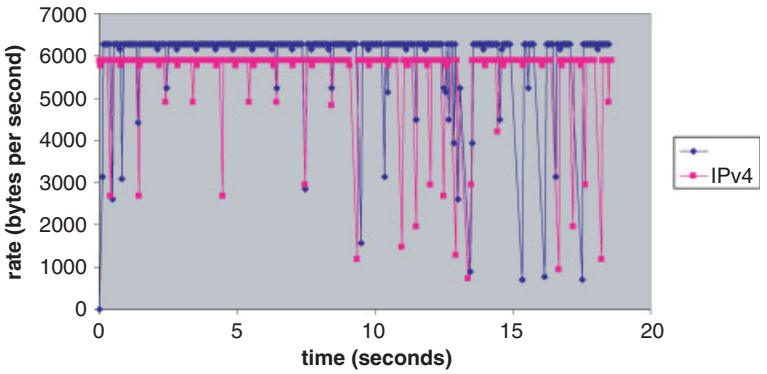


Figure 4. IPv4–IPv6 bandwidth consumption using G.711 A-Law.

codec (64 Kbps) because of the Silence Suppression function that was activated at the transmitting client.

10.2. Experiment 2: IPv4 communication with competing UDP traffic

This time we repeated the initial experiment, but we also added some background traffic to compete with our OpenH323 application at the bottleneck link. Our intention was to be able to compare the relative performance of the IPv4 and the IPv6 version of the application, in order to more thoroughly verify the proper operation of the IPv6-enabled application, and also to make some more general observations for applications that are ported to the new Internet Protocol. Since the bottleneck link was rather big compared to the demands of our application, we generated competing traffic that was more than an order of magnitude larger than the H.323 traffic. Although this fact makes it more difficult for us to obtain detailed results, since we have to take into account the relative weight of each type of traffic, we believe that this situation is closer to a typical scenario of a high bandwidth congested link. Our experiments model a broadband network, that is however, to a large degree congested because of heavy use of a lot of competing applications (like peer-to-peer networks or other multimedia streaming sources). Because OpenH323 uses UDP, we chose to also generate a UDP traffic stream, since more gentle TCP traffic would be significantly reduced by the UDP traffic. UDP is also more typical of the usual applications with high bandwidth demands like real-time applications that could compete with the H.323 client in a typical scenario (multimedia streaming applications, other videoconferencing applications, peer-to-peer applications, etc.).

As we can see at Figure 5, the competing traffic reduced the transmission rate of the H.323 traffic, and therefore also reduced at a large and visible extent the quality of the video received at the other endpoint.

The reduction at the transmission rate of the H.323 traffic was not constant. Instead, there were time periods when the H.323 traffic actually regained most of its initial bandwidth (close to 16 KBytes/s). This effect probably has to do with the fact that the H.323 traffic was relatively small compared to the artificially generated UDP traffic, and therefore minor variations at the

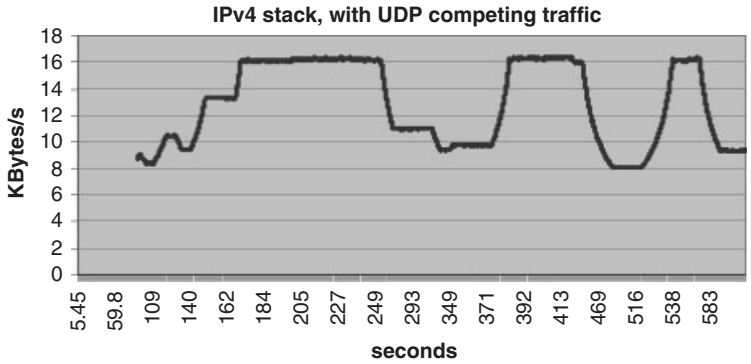


Figure 5. OpenPhone operation with IPv4 stack and UDP competing traffic.

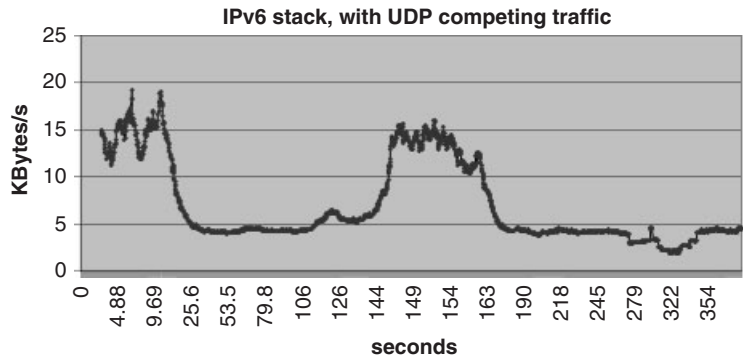


Figure 6. OpenPhone operation with IPv6 stack and UDP competing traffic.

generated traffic (perhaps because of processor or network stack limitations) reflected more heavily at the H.323 traffic.

10.3. Experiment 3: IPv6 communication with competing UDP traffic

When we repeated the above experiment using the IPv6 stack, the results were even more dramatic, because of the slightly bigger bandwidth consumption of IPv6. The transmission rate was significantly reduced, and so did the receiving video quality. The losses reported by RTCP were also 100% more than without the competing traffic (Figure 6).

We again observed the effect of a periodic effort by the H.323 traffic to regain more bandwidth, which we suspect is due to the same reasons as mentioned in the similar experiment conducted with the IPv4 stack. A concern also has to be the fact that the Windows 2000 IPv6 stack that was used for transmission is experimental, and is therefore probably not as optimized as the Windows 2000 IPv4 stack. This is also an observation that is reported in Reference [33], where the authors conduct detailed experiments using IPv6 stack for Windows and Solaris.

10.4. Experiment 4: IPv6 communication with competing TCP traffic

Our next experiment repeated the above described scenario, only that this time we chose the competing traffic to be carried by the TCP protocol, which is much more sensitive to congestion than UDP. This experiment models the scenario of an H.323 application competing with a lot of processes that occupy a lot more bandwidth than H.323 in total, but are using the TCP transport protocol, and are therefore more sensitive to congestion and the resulting packet losses.

The behaviour of the application again was in the range of 5–14 Kbps, although we observed some variations both in the transmitting rate and the reception quality of the video image, as shown in Figure 7. These variations are more intense than in the previous experiments. A reason for this behaviour can be the fact that the TCP protocol slowly tries to regain bandwidth that it has lost due to congestion through an AIMD (Additive Increase, Multiplicative Decrease) algorithm. When the artificially generated TCP traffic tried to increase its transmission rate, the resulting congestion caused more packets to be lost for the H.323 application. In total, RTCP reported a quite high 5.5% packet loss rate.

Figure 8 shows the impact that the H.323 application had on the competing TCP traffic. The transmission rate of the TCP traffic suffered almost a 30% decrease, since the

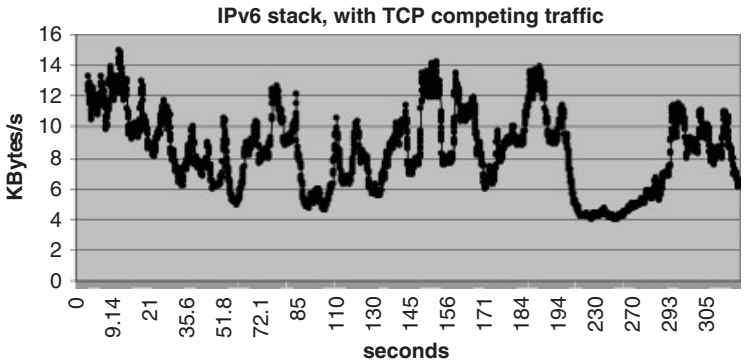


Figure 7. OpenPhone operation with IPv6 stack and TCP competing traffic.

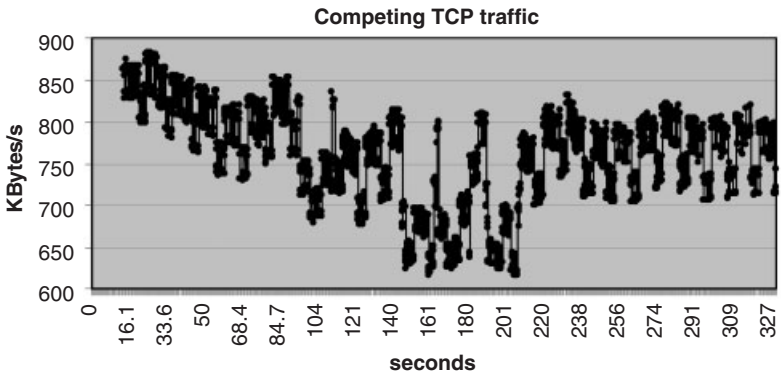


Figure 8. TCP traffic.

introduction of UDP traffic caused network congestion, from which TCP is unable to quickly recover.

Throughout the experiment, the TCP artificially generated traffic had a widely varying transmission rate, as it constantly tried to increase its bandwidth in a congested link. It is also worth noting that because of the 'pessimistic' operation of TCP, most of the time the competing traffic was far below the capacity of the link (after deducting the bandwidth that was consumed by the H.323 UDP traffic). This happened because each time TCP tried to increase the transmission rate, it soon leaded to congestion, and this in turn had the effect of aggressively (multiplicatively) decreasing the TCP transmission rate.

10.5. PESQ results

PESQ (perceptual evaluation of speech quality) [29] is a recommendation by ITU-T (P.862) that provides the standard for an objective method for end-to-end speech quality assessment of narrow-band telephone networks and speech codecs. It is a revision of the earlier P.861 standard (PSQM) in order to be more suitable for VoIP networks since in this case it will have to deal with higher distortions and delays. PESQ reports for each test a MOS (Mean Opinion Score) that evaluates the voice quality, that ranges from -0.5 to 4.5, although for most cases the output range is between 1.0 and 4.5, which is the range for an ACR (Absolute Category Rating) experiment.

The advantage of such an objective method for evaluating the voice quality is that we can easily acquire results that can be understood and evaluated without the cost and effort required for conducting a subjective test using trained people. For our PESQ testing, we have used version 1.2 of the reference implementation for P.862 from ITU-T.

Both the IPv4 and the IPv6 experiment with OpenPhone yielded very similar results regarding their PESQ MOS score: comparing the original IPv4 signal from our first audio experiment with the degraded one after it had been received at the other endpoint, we got a MOS at 2.296, while the same comparison for the IPv6 signal resulted at a 2.253 MOS score. Further repetitions of the evaluation produced results very close to the ones just reported (the difference was within 5% of the MOS value). These scores can be characterized as poor to fair, which is a reasonable result for a VoIP application in a production environment.

11. CONCLUSIONS

The number of required IP addresses for the near future (in order to address all the hosts and embedded systems) is expected to rise to the order of billions. IPv6 can provide this huge number of IP addresses, in addition to providing benefits like auto-configuration capabilities and QoS support. The larger IPv6 header introduces nevertheless some additional overhead, which is more significant for low-rate applications. In order to evaluate an IPv6 application and compare it with an equivalent IPv4, a number of experiments with various scenarios can prove useful.

The experience we gained from our efforts shows that porting applications to IPv6 will play a crucial role to the further adoption of the new Internet Protocol. While for many applications porting is going to be straightforward, for projects like OpenH323 that

have developed a large code base with low-level functionality a lot more effort is going to be required. The original structure of the code, as well as the language chosen for the implementation play a crucial role. The authors in Reference [12], for example, were able to focus on a more high-level approach of the porting procedure, because the newest versions of Java, unlike C++, transparently support IPv6. In the latter case, the preferred approach is the use of an object-oriented software architecture based on a class able to manage simultaneously IPv4 and IPv6 network connectivity. This architecture makes it possible to develop applications for IPv4 which require very few modifications to also simultaneously work on IPv6. The further development of automatic tools that assist the porting procedure is essential, and will probably have to be directed in two ways: supporting more languages than C\C++ in order to be useful for more programmers, and becoming more intelligent in helping the programmer with more subtle issues.

We have verified the correct operation of the OpenH323-based applications over IPv4 and IPv6 by conducting experiments with audio, video with and without competing traffic. We have also reinforced our subjective observations by using PESQ to objectively evaluate the quality for both cases. The results from our experiments clearly demonstrate the need for some sort of QoS mechanisms that will be able to compensate for the loss of quality that we observe when there is a congested link, especially when the competing traffic is UDP-style. The IPv4 and IPv6 versions behave roughly the same, although the slightly larger overhead of IPv6 due to the larger standard header makes the IPv6 version a bit more sensitive to congestion. Since a large part of the traffic in modern and future networks can be expected to be UDP, non-backtracking traffic, applications that are sensitive to congestion, like real-time applications, will need some kind of support from the network. This could be achieved through the use of QoS mechanisms and predefined service agreements. The different results in our experiments also demonstrate that in order to design and experimentally test the performance and efficiency of these mechanisms, a realistic and balanced assessment of the common patterns for Internet traffic is necessary [34]. Further research on the issue of the impact of QoS mechanisms on the performance of the real-time applications can be found in Reference [6], where these mechanisms are designed, implemented and evaluated on a number of experiments, including the usage of real-time applications based on the H.323 protocol.

12. FUTURE WORK

Our future work includes the extension of the above-mentioned experiments to greater actual IPv6 networks, and in particular using the experimental IPv6 network of the 6NET project [14]. We also intend to expand the collection of tested IPv6 stacks and comparatively evaluate their performance and maturity with regard to the transmissions by the videoconferencing applications. Moreover, we intend to investigate possible benefits in the area of quality of service (QoS) by using the traffic class and flow label fields in the IPv6 header, and the benefits in the area of security by using the Authentication Header and the IP encapsulating security payload (ESP). In the area of configuring the QoS mechanisms, we intend to further experiment and investigate the proper configuration parameters in order to optimize the performance of various traffic classes.

ACKNOWLEDGEMENTS

We want to thank the European Commission (EC) for supporting this work in the context of 6NET IST project (Contract number: IST-2001-32603). In addition, we want to thank our partners in the 6NET project for their valuable co-operation. More information regarding 6NET project can be found in http://www.6net.org.

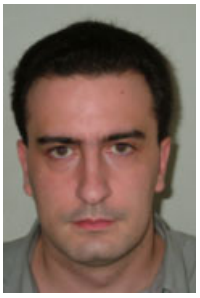
REFERENCES

1. Deering S, Hinden R. Internet Protocol, Version 6 (IPv6) Specification. Internet Engineering Task Force RFC 2460, December 1998.
2. Shin M, Hong Y, Lee SJ, Lee J, Kim Y. Application aspects of IPv6 transition. http://www.ietf.org/internet-drafts/draft-shin-ngtrans-application-transition-01.txt
3. Blanchet M, Cormier A, Parent F. Porting applications to IPv6: simple and easy!, May 2000. http://www.viagenie.qc.ca/en/ipv6/presentations/
4. Staszkiewicz CP. Porting large scale infrastructure applications to IPv6. *German IPv6 Summit 2004*, Bad Godesberg, 29 June–1 July 2004.
5. Bouras C, Gkamas A, Stamos K. From IPv4 to IPv6: the case of OpenH323 library. *SAINT 2003*, Orlando, Florida, 27–31 January 2003; 196–199.
6. Bouras C, Gkamas A, Primpas D, Stamos K. Quality of Service aspects in an IPv6 domain. *2004 International Symposium on Performance Evaluation of Computer and Telecommunication Systems* (*SPECTS' 04*), San Jose, California, U.S.A., 25–29 July 2004; 238–245.
7. Bouras C, Gkamas A, Josset S, Stamos K. Adding IPv6 support to H.323: Gnomemeeting/openH323 port. *11th International Conference on Software, Telecommunications and Computer Networks* (*SoftCOM 2003*), Croatia, Italy, 7–10 October 2003.
8. Microsoft Corporation, Adding IPv6 capability to Windows Socket Applications.
9. Sun Microsystems, Porting Networking Applications to the IPv6 APIs.
10. Stevens WR. *Network Programming*, vol. 1 (2nd edn). Prentice-Hall: Englewood Cliffs, NJ, 1998.
11. Guide to DIGITAL UNIX IPv6. http://www.ipv6.zk3-x.dec.com/userguide/TITLE.HTM
12. Ortiz R, Robles T, Salvachua J. Porting the session initiation protocol to IPv6. *IEEE Internet Computing* 2003; **May–June**:43–50.
13. Ettikan K, Chong TW. Portability issues for IPv4 to IPv6 applications. *APRICOT 2001*, Kuala Lumpur, Malaysia, 26 February–2 March 2001.
14. 6NET project. http://www.6net.org/
15. Euro6IX project. http://www.euro6ix.net/
16. 6INIT project. http://www.6init.org/
17. KAME project. http://www.kame.net/
18. O'Hanlon P, Jiang S, Kirstein P. IPv6 Globus—Experiences, GGF8, 26 June 2003. http://www.6net.org/publications/presentations/hanlon-globus.pdf
19. Libpnet6. http://pnet6.sourceforge.net/
20. OpenH323 project. http://sourceforge.net/projects/openh323
21. Packetizer, H.323 information site. http://www.packetizer.com/iptel/h323/
22. GnomeMeeting. http://www.gnomemeeting.org/
23. Microsoft IPv6 Technology Preview for Windows 2000. http://msdn.microsoft.com/downloads/sdks/platform/tpipv6.asp
24. Sun downloads. http://www.sun.com/download/index.jsp?cat=Operating%20Systems&tab=3
25. Compaq IPv6 Porting Assistant. http://www.tru64unix.compaq.com/internet/ipv6portingassistant/
26. MSDN Library, Windows Sockets Version 2.
27. Gilligan R, Thomson S, Bound J, McCann J, Stevens W. Basic socket interface extensions for IPv6. Internet Engineering Task Force RFC 3493, February 2003.
28. Sommerville I. *Software Engineering* (5th edn). Addison-Wesley: Reading, MA, 1995.
29. www.pesq.org
30. JNDmetrix technology, Sarnoff Corporation.
31. Iperf traffic generator. http://dast.nlanr.net/Projects/Iperf/
32. Sniff'em network analyzer. http://www.sniff-em.com/
33. Zeadally S, Raicu I. Evaluating IPv6 on Windows and Solaris. *IEEE Internet Computing* 2003; **May–June**: 51–57.
34. Thomson K, Miller GJ, Wilder R. Wide area internet traffic patterns and characteristics. *IEEE/ACM Transactions on Networking* 1997; **11**(6): 10–23.

AUTHORS' BIOGRAPHIES

Dr Christos Bouras obtained his Diploma and PhD from the Computer Science and Engineering Department of Patras University (Greece). He is currently an Associate Professor in the above department. Also he is a scientific advisor of Research Unit 6 in Research Academic Computer Technology Institute (CTI), Patras, Greece. His research interests include Analysis of Performance of Networking and Computer Systems, Computer Networks and Protocols, Telematics and New Services, QoS and Pricing for Networks and Services, e-learning, Networked Virtual Environments and WWW Issues. He has extended professional experience in Design and Analysis of Networks, Protocols, Telematics and New Services. He has published 200 papers in various well-known refereed conferences and journals. He is a co-author of 7 books in Greek. He has been a PC member and referee in various international journals and conferences. He has participated in R&D projects such as RACE, ESPRIT, TELEMATICS, EDUCATIONAL MULTIMEDIA, ISPO, EMPLOYMENT, ADAPT, STRIDE, EUROFORM, IST, GROWTH and others. Also he is member of, experts in the Greek Research and Technology Network (GRNET), Advisory Committee Member to the World Wide Web Consortium (W3C), IEEE Learning Technology Task Force, IEEE Technical Community for Services Computing WG 3.3 Research on Education Applications of Information Technologies and W 6.4 Internet Applications Engineering of IFIP, Task Force for Broadband Access in Greece, ACM, IEEE, EDEN, AACE and New York Academy of Sciences.

Dr Apostolos Gkamas obtained his Diploma, Master Degree and PhD from the Computer Engineering and Informatics Department of Patras University (Greece). He is currently an R&D Computer Engineer at the Research Unit 6 of the Research Academic Computer Technology Institute, Patras, Greece. His research interests include Computer Networks, Telematics, Distributed Systems, Multimedia and Hypermedia. More particular he is engaged in transmission of multimedia data over networks and multicast congestion control. He has published 9 papers in international Journals and 28 papers in well-known refereed conferences. He is also co-author of two books (one with subject Multimedia and Computer Networks and one with subject Special Network Issues). He has participated in R&D projects such as OSYDD, RTS-GUNET, ODL-UP, ODL-OTE, TODAY'S STORIES, ATMNet, 'Technical Consultant to Pedagogical Institute', ELECTRA and Teleteaching Service for Greek PTT and is currently involved in projects IPv6—GRNET, ASP-NG (ASP-New Game—IST-2001-35354) and 6NET (LARGE-SCALE INTERNATIONAL IPv6 TESTBED—IST-2001-32603).



Primpas Dimitrios was born in Sparti, Greece in 1980. He obtained his diploma from Computer Engineering and Informatics Department of the Polytechnic School of the University of Patras on November 2002. Next, he was accepted in the postgraduate program 'Computer Science and Technology' in the same department and on April 2004 he obtained his Master Degree. Now he continues the postgraduate studies in the same department to receive PhD. He works in the Research Unit 6 of Research Academic Computer Technology Institute and on Telematics, Distributed Systems and Basic Services Laboratory of the Computer Engineering & Informatics Department, in Patras University. His interests include: networks, protocols, Quality of Service, Managed bandwidth services and Network applications. He had worked on GRNET-IP-MBS project and currently works on 6NET (LARGE-SCALE INTERNATIONAL IPv6 TESTBED—IST-2001-32603), ASP-NG (ASP-New Game—IST-2001-35354), Advanced Services GRNET II/VNOC 2 and GN2 projects.

Kostas Stamos was born in Patras, Greece in 1978. In 1996 he entered the Computer Engineering and Informatics Department at the University of Patras. In 2001 he completed his diploma thesis on the subject of multicast video transmission supporting adaptive QoS. Graduated in September 2001 with G.P.A 8.65/10. Since October 2001 he is a student at the graduate program of the Computer Engineering and Informatics Department at the University of Patras, where in March 2003 he obtained his Master Degree.

He has worked for the Networking Technologies Sector of CTI from the end of 1999 until December 2000, where he participated at the implementation of the web site www.sch.gr, was involved in various maintenance and management tasks for the project 'Askoi toy Aioloy' and the development of a webmail service.

Since July 2001 he works as a R&D Computer Engineer with Research Unit 6 of CTI, where he has participated in R&D projects like GN-2, ASP-NG, 6NET, ATMNET and the development of videoconferencing applications for the Greek Telecommunications Organization (OTE).

He has published 3 articles in Journals and 14 papers in well-known refereed conferences.He owns Cambridge Proficiency in English and Kleines Sprachdiplom in German.

