# Performance Evaluation of an IPv6-capable H323 Application[1]

C. Bouras, A. Gkamas, D. Primpas, K. Stamos
*Research Academic Computer Technology Institute*
*Riga Feraiou 61, GR-26221 Patras, Greece*
*and*
*Computer Engineering and Informatics Department*
*University of Patras*
*GR-26500 Patras, Greece*
*Tel:+30-2610-{960375, 960465, 996954, 960316 }*
*Fax:+30-2610-{996314, 960358, 960358, 960358}*
*e-mail: {bouras, gkamas, primpas, stamos}@cti.gr*

## Abstract

*OpenH323 is an open source H.323 implementation that has been ported to IPv6. In this paper we briefly introduce the library architecture and the performance citeria with which the ported version should be evaluated. 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.*

## 1. Introduction

The new version of IP, IPv6 [9], constitutes an effort to overcome the inborn limitations of IPv4, in order for the new protocol to be able to respond to the new needs as they shape today in the Internet. More than simply increasing the address space, IPv6 offers improvements like built-in security support, plug and play support, no checksum at the IP header and more flexibility and extensibility than IPv4. IPv6 also facilitates efficient renumbering of sites by explicitly supporting multiple addresses on an interface. The widespread adoption of the new Internet Protocol will fuel innovation and make possible the creation of many new networking applications. It will also allow the replacement of the NAT solutions that have been implemented today in order to workaround the lack of IPv4 addresses. NAT introduces a number of problems to network applications that need knowledge of the IP address of the host machine or want to take advantage of Quality of Service mechanisms, like VoIP implementations.

The transition phase from IPv4 to IPv6 has raised many discussions among the Internet community. 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. Unfortunately, the vast majority of network applications in existence today, and especially multimedia applications, presume the use of the IPv4 protocol, so a transition to IPv6 will have 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. For this reason, we decided to port to IPv6 the library upon which the OpenH323 project is based, a large open-source library [1]. This way, we are able to use a wide range of real-time applications over IPv6 and experiment with their performance. It is also interesting to investigate the behaviour of IPv6 applications using QoS mechanisms, since they promise to effectively serve real-time applications in high bandwidth networks.

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 [10] focuses on Windows applications, but at the same time offers some general guidelines that apply to any application for any operating system. In [13], the authors emphasize more on some general knowledge that a programmer must acquire before dealing with the problem of porting applications to IPv6. There are also books [12]

[1]This work was partially supported by the 6NET project founded by the IST program of European Commission (IST contract No: 2001-32603) [5]

that can provide useful assistance to a programmer on this task. Recently, a paper by Robles, Ortiz and Salvachua presented the authors' results from porting a SIP implementation to IPv6 [11].

The rest of the paper is organised as follows: Section 2 gives a short introduction to the OpenH323 open source implementation of the H.323 standard. In section 3 we explain what we can expect from the experiments and how we should evaluate them. Section 4 presents the testbed we used for running the experiments, and section 5 provides the details of the results of our experiments and the conclusions we draw from each one. Finally, section 6 presents our final conclusions and our planned future work.

## 2. OpenH323 project

The OpenH323 project [4] develops an open source implementation of the H.323 standard in the form of a central library, the OpenH323 library, which is also based on another open source library called PWLib. The open source OpenH323 library can be used for the rapid development of applications that wish to use the H.323 protocol 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 H323 connection, various types of H323 channels, gatekeeper and transport protocols.

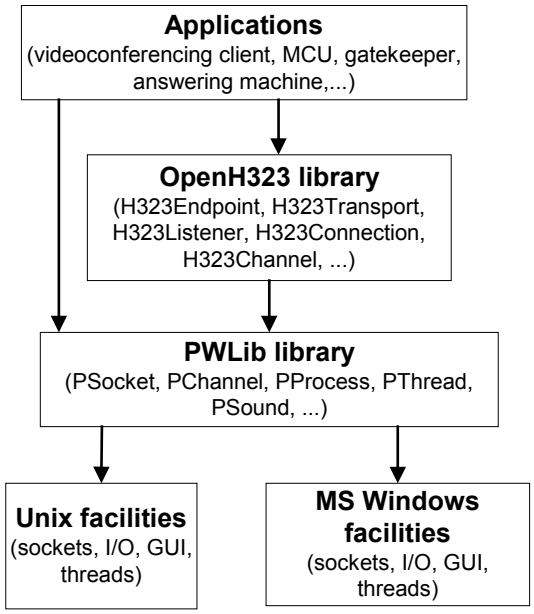


**Figure 1. Relationship of the OpenH323 and PWLib libraries**

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, a graphical H.323 client for Linux. Most of these applications require little additional effort in order to be used over IPv6 since the base libraries have been ported, thereby giving us the opportunity to test the IPv6 stack in various scenarios.

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.

The OpenH323 project and most of the applications it supports have now been ported to IPv6 [1].

## 3. Performance & Evaluation Criteria

There are various parameters that have to be examined when an application has been ported to IPv6, that determine the quality and the usefulness of the ported application. These parameters include (but are not limited to) whether the application simultaneously supports IPv4 and IPv6, compatibility with all the IPv6-related RFCs, capability to work with multiple DNS results (because of the IPv4-IPv6 coexistence), use of multicast or anycast addresses, and the level of support for new IPv6 features like the Flow Label for QoS schemes. In this work however, we are mainly concerned with the evaluation of the application regarding its performance and bandwidth consumption characteristics, and in particular how they compare between IPv4 and IPv6. More detailed information on the above issues can be found at [1], [2].

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.

In the experiments we conducted we were especially careful to transmit exactly the same source in all of the experiments, so that possible differences in behaviour due to different media streams could be eliminated.

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 the first case, the administrator will probably have to maintain two different versions for the same application, since IPv4 and IPv6 are predicted to co-exist for a long time, and keeping and updating two versions of the same application doubles the required administrative and maintenance efforts.

Finally, 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 IPv-6 enabled application 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 1.

**Table 1. 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 |

## 4. Experiments Setup

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.

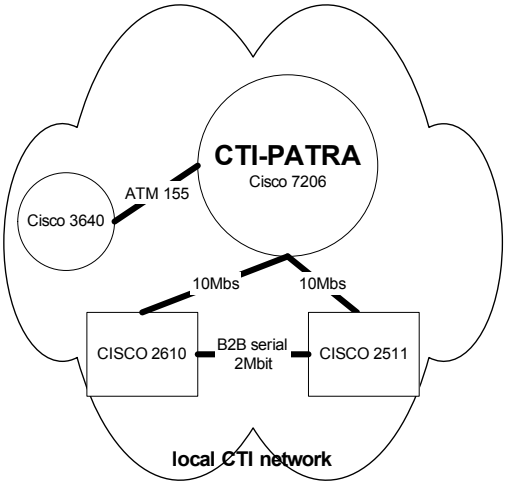


**Figure 2. Internal testbed for running the tests**

In order to create background traffic, we used the Iperf tool ([6]), 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 Ethereal and Sniff'Em network monitoring tools ([7], [8]).

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 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 64Kbps.

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.

The general purpose of our experiments is to evaluate the IPv6 version of the OpenH323 library, and particularly in comparison with the IPv4 version. Also, we want to observe how both versions behave when a link in the transmission path is congested because of the simultaneous transmission of other traffic. Finally, we test with the competing traffic being both UDP and TCP, because of the different behaviour characteristics of the two transport protocols, and the different impact that they have on the rest of the applications that use the same network links.

## 5. Experiments and Analysis

### 5.1. 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 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 at Figure 3, we obtained a steady transmit rate of 16 KBytes per second throughout the experiment. The quality of the video transmitted was relatively low, because of the characteristics of the H.261 codec.

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 OpenH323 protocol stack on a non-congested network using IPv6.

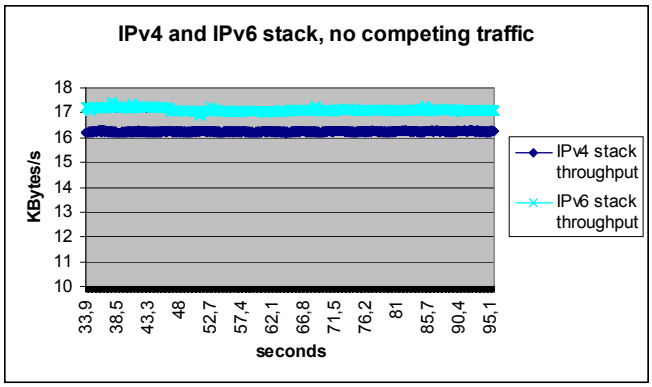


**Figure 3. OpenPhone operation without competing traffic**

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 per second, 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 can not however expect that this will always be the case. A transmission rate of around 140 Kbits per second 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.

### 5.2. 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. 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 UDP traffic, 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.

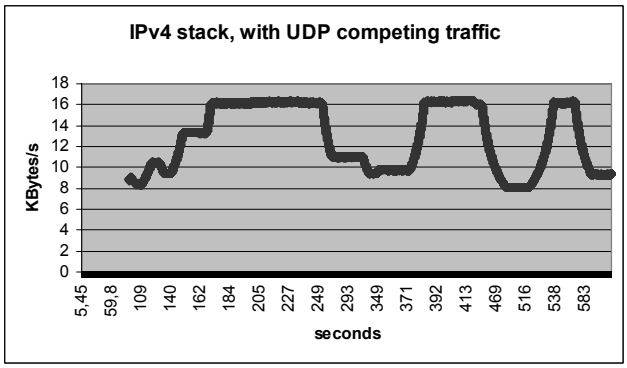


**Figure 4. OpenPhone operation with IPv4 stack and UDP competing traffic**

As we can see at Figure 4, 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 per second). 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 generated traffic (perhaps because of processor or network stack limitations) reflected more heavily at the H.323 traffic.

### 5.3. 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.

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.

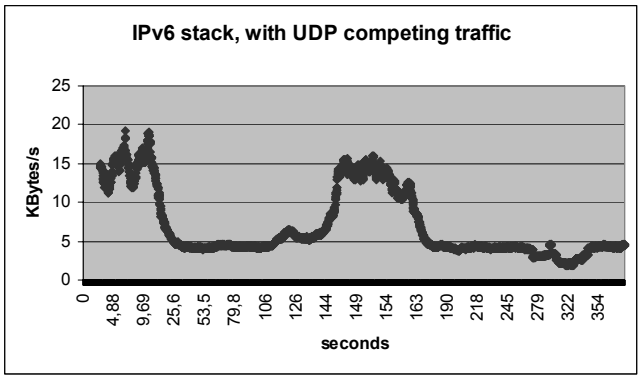


**Figure 5. OpenPhone operation with IPv6 stack and UDP competing traffic**

### 5.4. 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.

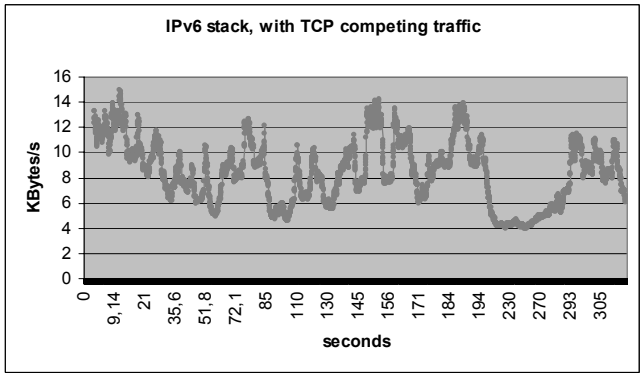


**Figure 6. OpenPhone operation with IPv6 stack and TCP competing traffic**

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 6. 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.

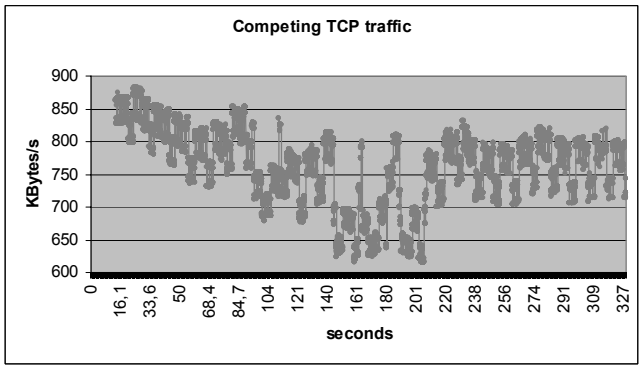


**Figure 7. TCP traffic**

Figure 7 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 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.

## 6. Conclusion and Future Work

The above results 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 safely 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.

At the next stages, we plan to repeat and expand the above described trials outside the CTI internal network, with other parties in the Greek IPv6 network and with parties outside Greece in order to investigate the operation of OpenH323 platform for WAN communication with many more hops. We will compare those results with the result acquired during the trial in our internal network. Furthermore, we plan to investigate the behaviour and the outcome of the trials using QoS DiffServ mechanisms like the gold service, that will benefit the OpenH323 traffic compared to the rest of the background traffic.

## 7. References


[1] C. Bouras, A. Gkamas, K. Stamos, “From IPv4 to IPv6: The case of OpenH323 Library”, SAINT 2003, Orlando, Florida, 27-31 January 2003, pp. 196-199

[2] C. Bouras, A. Gkamas, A. Karaliotas, D. Primpas, K. Stamos, “Issues for the performance monitoring of an open source H.323 implementation ported to IPv6-enabled networks with QoS characteristics” The 2003 International Conference in Internet Computing (IC 2003), Las Vegas, Nevada, USA, June 23 - 26 2003, pp. 765 - 771

[3] K. Thomson, G.J. Miller and R. Wilder, “Wide Area Internet Traffic Patterns and Characteristics” in IEEE/ACM Transactions on Networking, pp 10-23, 1997

[4] OpenH323 project, http://www.openh323.org

[5] 6NET project, http://www.sixnet.org

[6] http://dast.nlanr.net/Projects/Iperf

[7] http://www.sniff-em.com/

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

[9] S. Deering and R. Hinden, Internet Protocol, Version 6 (IPv6) Specification, Internet Engineering Task Force RFC 2460, December 1998

[10] Microsoft Corporation, Adding IPv6 capability to Windows Socket Applications

[11] R. Ortiz, T. Robles and J. Salvachua, Porting the Session Initiation Protocol to IPv6, IEEE Internet Computing, May – June 2003, pp. 43-50

[12] W. R. Stevens, Network Programming, Volume 1, 2nd Edition

[13] Sun Microsystems, Porting Networking Applications to the IPv6 APIs