Issues for the performance monitoring of an open source H.323 implementation ported to IPv6-enabled networks with QoS characteristics

Ch. Bouras A. Gkamas A. Karaliotas 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, 960355, 960440, 996182, 960316 }

Fax:+30-2610-{996314, 960358, 960358, 960358, 960358}
e-mail: {bouras, gkamas, karaliot, primpas, stamos}@cti.gr

Abstract

OpenH323 is an open source H.323 implementation that has been ported to IPv6. It therefore presents an opportunity to perform experiments and compare the performance of an H.323 application when using the IPv4 stack and when using the IPv6 stack. In this paper we initially introduce briefly the porting procedure and the methodology we used in order to achieve it. We then present the appropriate experiments that have to be performed 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 and the conclusions that they can lead us to.

Keywords: IPv6, OpenH323, Real Time Applications, Videoconferencing, QoS

1. Introduction

The new version of IP, IPv6 ([3]), 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. 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: (1) IPv6 has built in security support. (2) IPv6 eliminates the checksum from the IP header. (3) IPv6 is more flexible and extensible than IPv4. (4) IPv6 facilitates efficient renumbering of sites by explicitly supporting multiple addresses on an interface. (5) IPv6 supports plug and play operation.

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. 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. 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], [2]). 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 behavior of IPv6 applications using QoS mechanisms, since they promise to effectively serve real-time applications in high bandwidth networks. A number of research projects (6NET [10], Euro6IX [12], 6INIT [11], KAME [13]) are actively investigating the migration effort, the benefits from IPv6 and the performance issues involved, and have shared or are going to share their valuable experiences. Research Academic Computer Technology Institute (CTI) is one of the participants of the 6NET project, and this work was partially supported by the 6NET project founded by the IST program of European Commission (IST contract No: 2001-32603) ([10]).

This rest of the paper is organised as follows: Section 2 is an introduction to the H.323 protocol and its open-source OpenH323 implementation, and describes the methodology we used in order to make the OpenH323 project IPv6-enabled. Sections 3 and 4 present the evaluation and performance criteria that can apply to an IPv6 ported application. The experiments that we have planned in order to apply these criteria are described in Section 5. Finally, Section 6 summarizes and concludes the paper.

2. H.323 standard and OpenH323 library

H.323 ([6]) is an ITU recommendation, which defines a network architecture and the associated protocols necessary to voice and multi-media calls establishment. H.323 is defined for a packet-based network, and does not impose any network protocol, which can as well be IPv4 as IPv6 or IPX. The main entities of an H.323 based video network are the following: (1) End points: These are the H.323 clients, which are used by the end users. They can propose phone, video, fax, and application sharing functionalities. (2) Gateways: Gateways can be used for the interconnection between different networks (for example an IP phone network and a traditional phone network). (3) Gatekeepers: Gatekeeper makes it possible to be freed from the knowledge of called party IP address. It is then possible to call someone by his name. The gatekeeper is also able to manage the billing, and call filtering/authorization. (4) Multipoint Control Units (MCUs): A Multipoint Control Unit (MCU) makes it possible to manage a conference of more than two end points. Each user connects to the MCU and then is able to discuss with all the other connected people.

The OpenH323 project ([4]) 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 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. 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. 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 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.

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 ([5]), a graphical H.323 client for Unix. 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.

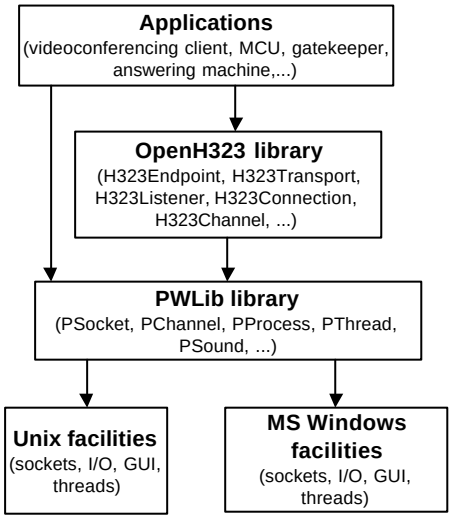

Figure 1 Relationship of the OpenH323 and PWLib libraries

Below we present in a step-by-step fashion the methodology that we used in order to port the OpenH323 and PWlib libraries to IPv6. It is intended as a guideline for similar projects that deal with porting a similarly large library to IPv6: (1) Study and understand the source code, highlighting the points where a change in the program's logic is probably necessary. (2) Parse the source code with an automatic tool like Checkv4.exe ([7]). (3) Modify the source code lines reported by the automatic tool, which are probably going to be rather straightforward. (4) Make any other necessary modifications in more subtle places not reported by the automatic tool. (5) Test and debug the code, correcting any issues that arise. (6) Verify completeness of porting effort.

Because of the huge size of the code base of the OpenH323 and PWLib libraries, a thorough examination line-by-line would be impossible, and therefore 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 ([7]), which is offered as part of the experimental IPv6 stack for Windows 2000. A problem 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. The OpenH323 library 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 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.

In general, there are a number of testing strategies that we followed: (1) 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. (2) Low-level testing: The opposite approach is to try and isolate specific classes and methods and try to test their behavior by using simple test applications with limited functionality. This way, errors can be more easily identified and their origin can be more easily attributed. (3) 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 one (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.

3. Evaluation of an application ported to IPv6

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 porting. Specifically, we identify the following criteria: (1) Ability to work with IPv6: Obviously, the application will have to seamlessly work with the IPv6 protocol. (2) IPv6 features involved: It is beneficiary if the application can make use of the new enhanced features of IPv6, so that these features can be examined and evaluated. For example, IPv6 defines the Flow id label that can be used in order to implement some QoS scheme. (3) Dependency on IPv4: Following a long-term approach, the application should not be dependent on any IPv4 aspect (e.g. the need for IPv4 DNS or IPv4 LDAP). (4) IPv4-IPv6 simultaneous support: Since IPv4 is going to co-exist with IPv6 for a long time, it is preferable for system administrators and application developers to have a single version to maintain, that is able to simultaneously support both IP protocols. (5) RFC compatibility: Conformance with all IPv6 RFCs, like for example RFC 2732 that defines the form literal addresses must have in URLs. (6) Dual-stack safe: The dual stack mechanism is going to be widely used for the foreseeable future, so an IPv6-enabled application has to be able to operate in a system with dual stack. (7) Multiple DNS: As described earlier, the DNS mechanism plays an important role for the communication between IPv4 and IPv6 hosts. Although some details are hidden from the application layer, in most cases it still has to be able to differentiate and handle each returned address by the DNS resolver properly. (8) Multicast, anycast: Apart from the unicast method of communication, IPv6 also makes use of multicast and introduces anycast. The multicast mechanism is especially useful for demanding real-time applications.

Moreover, 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 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. 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.

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. Performance Criteria

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. 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.

We performed some initial experimentation with a ported OpenH323 application, and compared the bandwidth rate required for an IPv4 and an IPv6 call. The application was a simple H.323 VoIP application using the G.711 A-Law codec for voice transmission. The tests were performed on a 10 Mbps Ethernet LAN, and there was no competition for the VoIP application we used, so that we could compare the differences in the bandwidth consumed in order to achieve the same voice quality. As shown in Figure 2, IPv6 maintains a data rate at around 50 Kbps (6.2 KBytes per second), while IPv4 maintains a data rate at around 47 Kbps (5.8 KBytes per second), almost 7% lower. 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. We intend to further investigate this behavior, which makes the use of QoS mechanisms like the ones described in Section 5 even more appropriate.

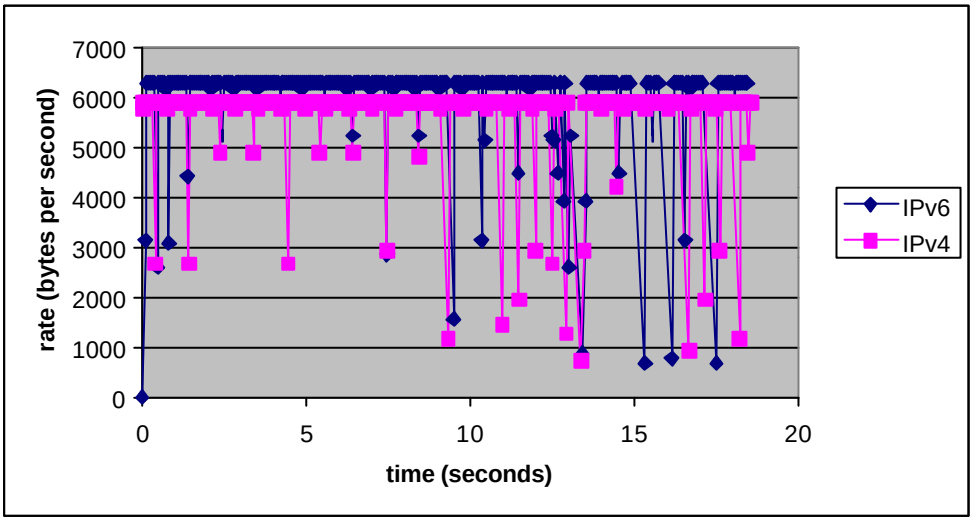

Figure 2: IPv4 – IPv6 bandwidth consumption

5. Planned Experiments

We plan to deploy our detailed experiments in a real environment in two stages. First, we are going to use the local experimental network testbed of CTI, and at a second stage we are going to use the infrastructure of 6NET, the largest European project on IPv6. Figure 3 presents the local CTI network, as well as the Greek part of the 6NET research network.

In order to evaluate the IPv6-compatible version of OpenH323, we have planned the

following experiments, which can be categorized in 3 main areas:

- Testing scenarios using best-effort: In these scenarios no QoS mechanism will be used. Apart from the OpenH323 traffic, we will use artificially generated traffic using a traffic generator, which can create and manage many simultaneous IP connections (TCP or UDP). It gives the opportunity to configure the TCP parameters such as TCP buffer size and TCP window size and allows the configuration of many parameters to the IP connections such as IP address, port number, protocol and packet size. We intend to fill the bi-directional links with background traffic, which will be treated with best effort service. The link utilization should be at or near 90% and the packet size should be 40 bytes, 552 bytes and 1500 bytes. The analogy for each size is 59%, 32% and 9%, in order to emulate real traffic as close as possible ([9]). The TCP and UDP packets analogy should be 70% and 30% respectively.
- Testing scenarios using QoS on gold service: This time the OpenH323 traffic will be treated using the gold service. The basic idea of gold service is that all the in profile packets must arrive to the destination, with the minimum possible delay and jitter. The out of profile packets may arrive or not, depending on the network's status. All the above parameters must have assigned to a service level agreement between the network provider and each client. This service can guarantee delay, jitter and packet loss. The mechanisms that will be used are the following: (1) Classification at the edge router according to the ports used by the OpenH323 application. (2) Policing at the edge router using the token bucket mechanism. The token bucket profile will be configured online and is a point for research. (3) Queue management. At each router there will be two queues, the first will serve the gold service and will be the priority queue. The second will serve the best effort traffic. The packets that are out of profile will be served as best effort traffic. (4) Finally, RED will be used at the best effort queue for congestion avoidance.
- Testing scenarios using QoS with more than one class of service: The QoS tests will then be continued, adding one more class of service. In particular, the additional class(es) will have worst treatment than the gold service but better than best effort. Its role will be to serve some packets better than best effort. The testing scenarios will be the same as above, with the difference that an additional traffic flow will be produced for that service. The mechanisms that are planned to be used, except from the corresponding for gold traffic are: (1) Classification at the router according to the port numbers. (2) Policing at the edge router using the token bucket algorithm. (3) Queue management. Three queues will be used: gold, best effort and one more, which will serve packets with more weight than the best effort. This step could be achieved with the use of Weighted Fair Queuing.

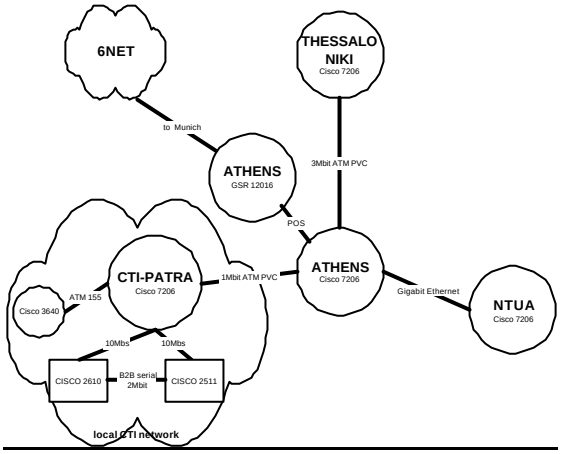

Figure 3: CTI testbed

For each scenario, we are going to compare the performance of the IPv6 OpenH323 application vs. the IPv4 application, in order to investigate the way the new Internet Protocol affects the network. In order to free ourselves from issues specific to one IPv6 or IPv4 implementation, we are going to use at least 2 different stacks for each scenario, Windows 2000 and Linux. The parameters we are going to measure for each experiment are packet loss, delay, jitter, throughput and bandwidth sharing with competing flows. In addition, the router

configuration in order to increase the performance of the OpenH323 application is an issue under investigation during the experiments.

6. Conclusions and future work

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 (if it exists), a number of experiments can prove useful.

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 ([10]). 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.

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] S. Josset, C. Bouras, A. Gkamas, K. Stamos, "Adding IPv6 support to H323: Gnomemeeting/openH323 port", IST Mobile & Wireless Communications Summit 2003, 15-18 June 2003, Aveiro – Portugal (submitted)

[3] Internet Protocol, Version 6 (IPv6) Specification - RFC 2460

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

[5] GnomeMeeting, http://www.gnomemeeting.org/

[6] Packetizer, H323 information site, http://www.packetizer.com/iptel/h323/

[7] Microsoft IPv6 Technology Preview for Windows 2000, http://msdn.microsoft.com/downloads/sdks/platform/tpipv6.asp

[8] CTI/RU6 OpenH323 porting project, http://ouranos.ceid.upatras.gr/openh323/

[9] 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

[10] 6NET project, http://www.sixnet.org

[11] 6INIT project, http://www.6init.org/presentations.html

[12] Euro6IX project, http://www.euro6ix.net

[13] KAME project, http://www.kame.net/