From IPv4 to IPv6: The Case of OpenH323 Library

Ch. Bouras[1,2] A. Gkamas[1,2] K. Stamos[1,2]
[1]Research Academic Computer Technology Institute, Riga Feraiou 61, GR-26221 Patras, Greece
[2]Computer Engineering and Informatics Dept., Univ. of Patras, GR-26500 Patras, Greece
Tel:+30-(0)610-{960375, 960355, 960316 }
Fax:+30-(0)610-{996314, 960358, 960358}
e-mail: {bouras, gkamas, stamos}@cti.gr

Abstract

In this paper, we discuss our experience form porting the OpenH323 platform to IPv6. We briefly discuss the structure of the platform, and we present the various problems we faced and choices we made, regarding the easiest approach to the porting procedure, the compatibility with earlier, IPv4-only versions of the platform, the existence of tools that could aid us with the porting task and the verification of result. We believe that the results of our effort can serve as guidelines for other similar projects and we present a general methodology which might be applicable to similar porting projects.

1. Introduction

The new version of IP, IPv6 [1], constitutes an effort to overcome the inborn limitations of IPv4, in order for the new protocol be able to respond to the new needs as they shape today in the Internet. However, the transition phase from IPv4 to IPv6 constitutes the main reason for the relative slow adoption of the new Internet Protocol: a lot of companies and network administrators are reluctant, facing what they perceive as a great challenge with large costs. Unfortunately, the vast majority of network applications in existence today 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.

The problem of porting existing applications to IPv6 has been so far addressed by several researchers. 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 [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. Furthermore, a number of research projects (6NET [12], Euro6IX [17], 6INIT [15], KAME [16]) are actively investigating the migration effort and the benefits from IPv6, and have shared or are going to share their valuable experiences. CTI (Research Academic Computer Technology Institute) is one of the participants of the 6NET project, and this work was partially supported by the 6NET project [12].

This rest of the paper is organised as follows: Section 2 gives a brief overview of the OpenH323 project structure. Section 3 presents the different ways of approaching the porting task. Section 4 summarizes the methodology we have followed during our porting efforts. Section 5 presents in detail the most significant modifications that had to be made to the source code. Section 6 addresses the problem of verifying when porting is correctly completed. In Section 7 we present our future work and in Section 8 our conclusions.

2. Structure of the OpenH323 library

The OpenH323 project [3] 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). It develops a central library, the open source OpenH323 library, which can be used for the rapid development of applications that wish to use the H.323 protocol [4] for multimedia communications over packet-based networks. It is written in 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, when the OpenH323 classes want to use an operating system mechanism, 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 and Unix systems, without modifying the source code.

The applications that have been developed on top of the OpenH323 library 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 Unix.

The above interactions can be seen in Figure 1.

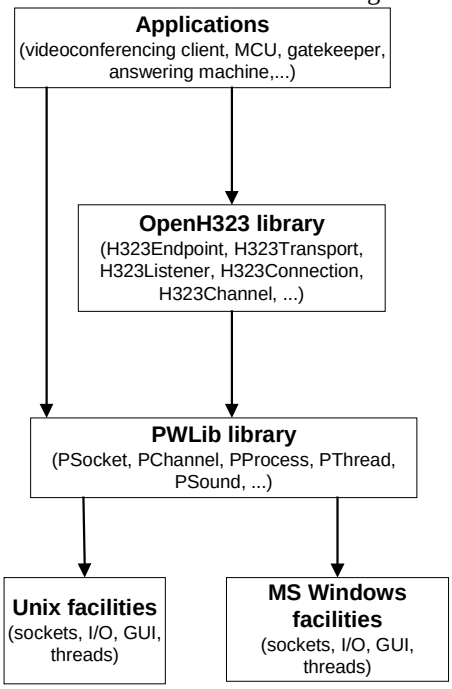

Figure 1 Relationship of the OpenH323 and PWLib libraries

3. 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. For the vast majority of applications, backwards compatibility with IPv4 will be needed, since there is going to be a long period of transition from IPv4 to IPv6. The alternatives for the porting task can be categorized as below:

- 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 that only works with IPv6. 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). This approach eliminates the complexities of having two source code bases or two different binaries, but 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. Therefore, in our case we 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.

4. Methodology

Below we present in a step-by-step fashion the methodology that we used during out porting efforts. It is intended as a guideline for similar projects that deal with porting a similarly large library to IPv6:

- Study and understand the source code, highlighting the points where a change in the program's logic is probably necessary.
- Parse the source code with an automatic tool like Checkv4.exe [5].
- Modify the source code lines reported by the automatic tool, which are probably going to be rather straightforward.
- 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. 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. Such a tool searches for patterns that are generally recognized as potential points that need modification, but in most cases there is still work to be done by the programmer after the lines suggested by the automatic tool have been modified. This phase depends greatly to 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.

5. Necessary modifications

In order to port the OpenH323 project we followed the methodology described in Section 4. There are a number of quite straightforward modifications that we had to carry 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.
- 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.
- Replacing hard-coded IPv4 addresses 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.

Because of the huge size of the code base of the OpenH323 and PWLib libraries, we used the automated Checkv4.exe tool by Microsoft [5], in order to trace down the most obvious IP protocol dependent points in the source code.

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. Unfortunately, the vast majority of network applications used today contain many hidden indirect IPv4 dependencies. which make porting a challenging undertaking.

This phase of testing and debugging was the longest and most difficult to complete. 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.

The most important problems that were revealed during the porting of the OpenH323 project that required changes in the logic of the source code and could not be easily identified from the beginning were the following:

- 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. This was by far the most time-consuming part of the modification phase.
- The socket API implementations of Linux and Windows are not fully compatible. Furthermore, the Windows IPv6 stack we used, which was included in the Microsoft IPv6 Technology Preview for Windows 2000 did not support some of features described in RFC 2553 [2] 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.
- A common phenomenon in network applications is the difference between various computer architectures in the order they store 2-byte values. (little endian or big endian). 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 8 2-byte fields.
- The size of the source code base (around half a million lines for both PWLib and OpenH323) is obviously a factor that makes the engagement with the project a challenging task.

6. Verifying porting completion

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. In smaller, single-purpose applications (and not libraries) 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. They could not be possibly tested using just a single application, and we extended the porting to include a wide range of OpenH323-based applications.

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:

- High-level testing: This testing strategy 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.
- 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.
- 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.

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. 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 challenging part was testing the PWLib library. Although many parts of its functionality were left totally unchanged, the wide range of functions offered by the heavily modified classes 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 inspection of the modified classes and methods.

7. Future work

In our future work, we intend to investigate the benefits that applications in general and the OpenH323 platform in particular can gain from the adoption of the IPv6 protocol, and its enhancements over IPv4. Furthermore, we intend to validate the feasibility of a version of the library that will be able to fully operate over both IPv4 and IPv6, and compare the different approaches to such a goal (two compiled versions, one version that differentiates on run-time, etc.).

8. Conclusions

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 further development of automatic tools 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.

9. References

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

[2] Basic Socket Interface Extensions for IPv6 – RFC 2553

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

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

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

[6] Compaq IPv6 Porting Assistant, http://www.tru64unix.compaq.com/internet/ipv6portingassistant/

[7] Solaris IPv6, http://wwws.sun.com/software/solaris/ipv6/

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

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

[10] "Network Programming, Volume 1", 2nd Edition, W. Richard Stevens

[11] Guide to DIGITAL UNIX IPv6, http://www.ipv6.zk3-x.dec.com/userguide/TITLE.HTM

[12] 6NET project, http://www.sixnet.org

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

[14] Application Aspects of IPv6 Transition, http://www.ietf.org/internet-drafts/draft-shin-ngtrans-application-transition-01.txt

[15] 6INIT project, http://www.6init.org/presentations.html

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

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