From IPv4 to IPv6: The Case of the OpenH323 Library

Authors: Ch. Bouras, A. Gkamas, K. Stamos

Presenter: Eri Giannaka

Research Academic Computer Technology Institute, Greece
University of Patras, Computer Engineering and Informatics Department, Greece

http://ru6.cti.gr

Introduction

- In this paper, we discuss our experience form porting the OpenH323 platform to IPv6
- We present the structure of the OpenH323 platform
- We discuss the problems we faced and choices we made during the porting procedure
- We discuss the compatibility with earlier, IPv4-only versions of the platform
- We investigate for the existence of tools that could aid us with the porting task
- We discuss the verification of result

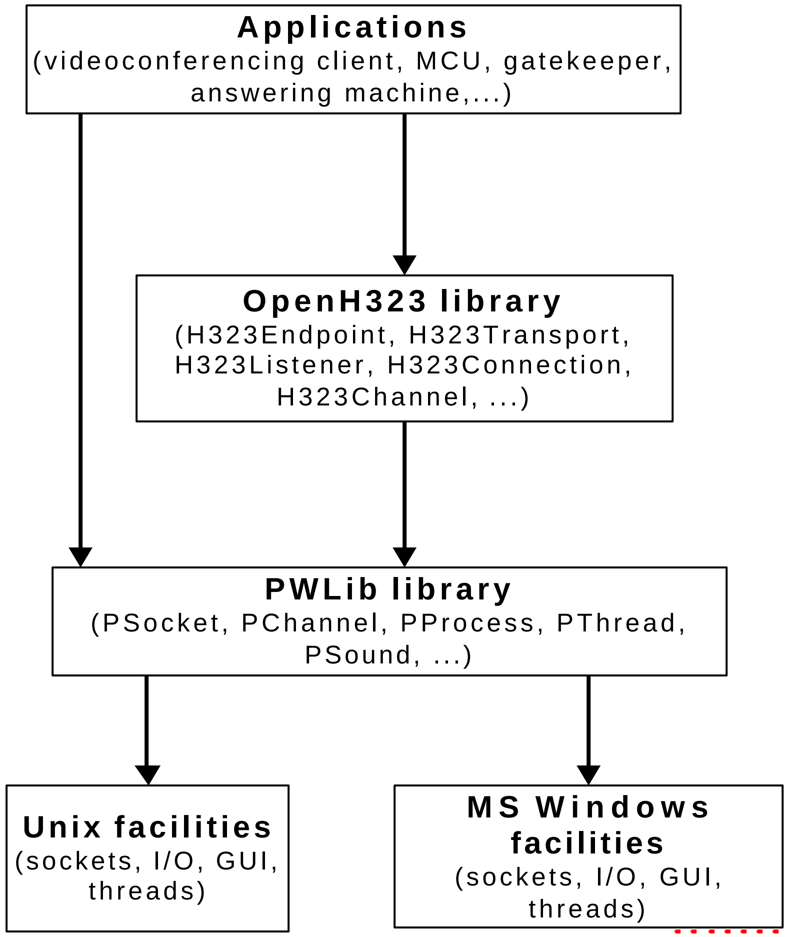
The Transition to IPv6

- The new version of IP, IPv6, 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
- The transition phase is an obstacle and the main reason for the slow adoption of IPv6
- 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 OpenH323 project

- H.323: A standard approved by the International Telecommunication Union (ITU) that defines how audiovisual conferencing data is transmitted across networks
- OpenH323: an open-source implementation of the H.323 protocol stack
- PWLib: open-source library that encapsulates I/O, GUI, multi-threading and networking functionalities, and implements basic “container” classes such as arrays, linear lists, sorted lists (RB Tree) and dictionaries (hash tables). 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

Structure of the OpenH323 project



Applications
(videoconferencing client, MCU, gatekeeper, answering machine,...)
OpenH323 library
(H323Endpoint, H323Transport, H323Listener, H323Connection, H323Channel, ...)
PWLib library
(PSocket, PChannel, PProcess, PThread, PSound, ...)
Unix facilities
(sockets, I/O, GUI, threads)
MS Windows facilities
(sockets, I/O, GUI, threads)

Size and scope of the OpenH323 project

- OpenH323 and PWLib are comprised of 500,000 source code lines in over 400 classes written in C++
- The applications that have been developed on top of the OpenH323 library include:
  - a command line H.323 client
  - H.323 videoconferencing server (MCU)
  - H.323 answering machine
  - H.323 gatekeeper
  - H.323 to PSTN and fax modem to T.38 gateways
  - GnomeMeeting, a graphical H.323 client for Unix

Porting OpenH323 to IPv6

- We believe that the OpenH323 project is a good candidate for porting to IPv6 for 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 allows us to deal with a large-scale project, and experience any difficulties and issues that might not arise in simpler cases
  - It presents a problem that can be approached by different angles, therefore allowing us to compare different strategies on the porting issue

Backwards compatibility with IPv4

- 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

	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

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

Supporting an IPv4 version

- Alternatives for porting:
  - Two source code bases, two binaries (IPv4-dependent and IPv6-dependent binaries)
    - Requires maintenance of two versions
  - Single source code base, two binaries (IPv4-dependent and IPv6-dependent binaries)
    - Using preprocessor directives we include either IPv4, or IPv6 compatible code
    - Two binaries exist in the system
  - Single source code base, single binary (IP version agnostic binary)
    - Capable of handling any type of IP protocol
    - Only one binary needs to be compiled and maintained
    - Can be tricky, requires more extensive changes

Methodology for porting procedure

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 Microsoft’s Checkv4.exe
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

Automated Tools

- 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
- Some of the tools of this kind available:
  - Microsoft’s Checkv4 for MS Windows
  - Sun’s Socket Scrubber for Solaris
  - Compaq’s IPv6 Porting Assistant for Tru64 Unix
- Most of these tools are free and work with C/C++

The most important straightforward modifications

- 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

Some problems we faced

- There were many parts in the code where the IP address was indirectly assumed to have a 4-byte length (in size of arrays, 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, (included in the Microsoft IPv6 Technology Preview for Windows 2000) did not support some of features described in RFC 2553 (Basic Socket Interface Extensions for IPv6) and has a small number of other differences
- 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 task challenging

Verifying porting completion – Testing strategies

- 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
- We used a combination of the above techniques, with emphasis to comparative testing

Conclusions – Future Work

- 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:
  - Better support for languages other than C¥C++
  - More intelligence in helping the programmer with subtler issues
- 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

Email info

Christos Bouras (bouras@cti.gr)

Apostolos Gkamas (gkamas@cti.gr)

Kostas Stamos (stamos@cti.gr)