An IP Next Generation Compliant *Java*™ *Virtual Machine*

Guillaume Chelius[1] and Éric Fleury[2]

[1] École Normale Supérieure de Lyon, `Guillaume.Chelius@ens-lyon.fr`
[2] LORIA, INRIA, `Eric.Fleury@loria.fr`

Abstract This paper discusses the implementation of an `IPv6.java.net` package allowing distributed applications to benefit from all the new aspects of the IPv6 protocol such as: security, flow control, mobility and multicast. Moreover, we present a full IP Next Generation Compliant *Java*™ *Virtual Machine*, that is a JVM which is able to handle both IPv4 and IPv6 protocols from standard Java APIs and in a fully transparent way for the users/programmers.

1 Introduction

Recently, there has been a strong interest in using heterogeneous, remote resources for high performance distributed computing. The concept of *Computational Grid* envisioned in Foster and Kesselman [11] has emerged to capture the vision of a network computing system that provides broad access not only to massive information resources but also to massive computational resources. At the same time, several other opportunities have also been highlighted, including the potential to realize time-shared, collaborative metacomputing and to exploit portability and code mobility through a language such as Java [4,12,13,16]. As a result, there have been many highly successful projects that focus on particular aspect of distributed computing and the networking part takes a major position, not only in the research fields but also in the deployment of ambitious projects propounding very large-scale metacomputing and the computational grid [10,14,20].

In the early nineties, in this context of technological improvement in both computer science and telecommunication areas, the expansion of the Internet has become such huge that the format of addresses was not adapted to the number of potential hosts any more. This was the main reason which pushed researchers to develop a new version of the IP protocol. In addition, this development was also a good way to introduce advances made in network research during the twenty five last years. These advances concern a wide domain covering security, flow control, mobility and multicast. All this work led to the conception of Internet protocol version 6 (IPv6 for short), also known as *IP next generation*. Much of the current IPv6 architecture has already been ratified, and implementations are emerging [9] even if – as an evolving technology – IPv6 is far from being a production network proposition at present. Several issues still remain.

In this paper, we address the advantages that distributed applications may benefit when using IPv6 and we present our implementation of an IPv6 Compliant *Java*™ *Virtual Machine*. One of the main reasons of the popularity of the Java programming language is its support for distributed computing [3]. Java's API for sockets, URL and other networking facilities is much simpler than what is offered by other programming languages like C or C++. Moreover, the Java$^{\text{TM}}$ Remote Method Invocation (RMI) was designed to make distributed application programming as simple as possible, by hiding the remoteness of distributed objects as much as possible. All these APIs that were developed and used with an IPv4 JVM will be fully available on our IPv6 Compliant *Java*$^{\text{TM}}$ *Virtual Machine*. This goal can be obtain by two complementary approaches: providing a `IPv6.java.net` package (API for IPv6) and/or running a JVM above an IPv6 stack and thus inherit all the enhancements of IPv6.

Section 2 contains a brief overview of IP next generation, and points out the main modifications from the IPv4 protocol that may present improvements for distributed applications. Section 3 presents the integration/implementation of IPv6 into Java, *i.e.*, an API supporting IPv6. Section 4 presents some extensions and improvement we add and results to validate our approach. Section 5 concludes with future research and software design directions.

2 A Quick overview of IPv6

The aim is to present briefly the main important differences between IPv4 and IPv6 that should benefit distributed applications in terms of improvement and new facilities. For a more precise description of IPv6, please refer to [6,15].

2.1 Addressing format

The ability to sustain continuous and uninterrupted growth of the Internet could be viewed as the major driving factor behind IPv6. IP address space depletion and the Internet routing system overload are some of the major obstacles that could preclude the growth of the Internet. Even though the current 32 bit IPv4 address structure can enumerate over 4 billion hosts on as many as 16.7 million networks, the actual address assignment efficiency is far less than that, even on a theoretical basis [15]. This inefficiency is exacerbated by the granularity of assignments using Class A, B and C addresses. By extending the size of the address field in the network layer header from 32 to 128 bits, IPv6 raised this theoretical limit to 2^{128} nodes. Therefore, IPv6 could solve the IP address space depletion problem for the foreseeable future.

2.2 Modifications of the IP stack protocols

IP. The modification of the address length is not the only change that occurred in the IP protocol. The aim behind all the modifications is to optimize the treatment inside a router in order to reduce the latency in each intermediate router along a packet's delivery path. The most important optimization is the simplification of the header format which has now a fixed size. Some IPv4 header fields (*e.g.*, checksum, fragmentation) have been dropped or made optional to reduce the common-case processing cost of packet handling and to keep the bandwidth overhead of the IPv6 header as low as possible in spite of the increased size of the addresses. IPv6 options are placed in separate headers that are located in packets between the IPv6 header and the transport-layer header. Since most IPv6 option headers are not examined or processed by any router along a packet's delivery path until it arrives at its final destination, this organization facilitates a major improvement in router performance for packets containing options. A key extensibility feature of IPv6 is the ability to encode, within an option, the action which a router or host should perform if the option is unknown. This permits the incremental deployment of additional functionalities into an operational network with a minimal danger of disruption.

ICMPv6. ICMPv6 is more than a simple evolution of ICMP. It now also supports the functionalities of ARP and IGMP. It still handles control or error messages like *unreachable destination*, *no route to host*, *packet too big* but it becomes also responsible for the *neighbor discovery* protocol. This protocol is used to perform different tasks: *Address Resolution*, *Neighbor Unreachability Detection*, some configurations and *Redirection Indication* and to support autoconfiguration.

TCP and UDP. Modifications made to TCP and UDP concern two main areas. The first one deals with data integrity: a pseudo IP header has been added to the computation of the checksum (it is now mandatory for UDP). The second one concerns packet length. It is now possible to send *jumbograms*, which are packets longer than 65536 bytes. They can notably be useful in distributed computation. For example, an IP over Myrinet will benefit from the introduction of jumbogram. Since there is no physical MTU fixed in Myrinet, packet can be of any size which allow to achieve efficient throughput.

2.3 Other new features

Support for authentication and privacy: IPsec. IPv6 includes the definition of an extension which provides support for authentication and data integrity. This extension is included as a basic element of IPv6 and support for it will be required in all implementations. IPv6 also includes the definition of an extension to support confidentiality by using encryption. Support for this extension will be strongly encouraged in all implementations. The use of IPsec in a computational grid context can be very useful in the deployment of sensitive applications that are potentially *griddable*. The fact that security is pushed to the network layer will increase the performance of such systems and allows an easy creation of VPN (*Virtual Private Network*) that can be deployed over the Internet.

Multicast. The scalability of multicast routing is improved by adding a ”scope” field to multicast addresses. Multicast provides more flexibility in a more efficient way and it is heavily used even in the most basic protocol like Neighbor Discovery Protocol (NDP). Therefore, multicast is mandatory in every IPv6 hosts/routers whereas it was optional in IPv4. The principle of multicast opens numerous perspectives in the domains of metacomputing or distributed computation. Indeed, it enables the dispatch of information and communication between several entities in a new and efficient way and appears to be a key point in Distributed Interactive Application (DIS) [17].

Mobility. The principle of mobility is quite new. The aim is that computers can stay connected to the Internet despite their physical moves. It assumes that a computer can automatically obtain an address when it is connected to an IP network but also that it is reachable on a constant address. The result is a great flexibility in network management and architecture. A protocol, based on the use of *extensions*, was written to negotiate such associations (between the *mother address* of a computer and the router where it is actually located). This new capability may allow new interfaces to be easily designed to support a range of scientific activities across distributed, heterogeneous computing platforms [18].

Quality of service capabilities. A new capability is added to enable the labeling of packets belonging to particular traffic ”flows” for which the sender has requested special handling, such as non-default quality of service or ”real-time” service. This traffic class will be useful to enhance distributed applications that have specific features (interactivity, hard time constraint, ...) like DIS [5]. A new type of address, called a ”cluster address” is defined to identify topological regions rather than individual nodes. The use of cluster addresses in conjunction with the IPv6 source route capability allows nodes additional control over the path their traffic takes.

3 Developing an IPv6 package for Java

The first step of our project, bringing IPv6 to Java, was to create a package, called `fr.loria.resedas.net6`, that provides the possibility to deal with IPv6 networking programming. This API has to be coherent with regard to the Java architecture and, in particular, with the existing network programming API as described in [8].

3.1 Architecture

If we look at the different classes of the *java.net* package, three main levels can be discerned. The first one is composed of classes which only deal with Java mechanisms: the *exceptions*. The second one is composed of classes which implement session or application mechanisms. Finally, the third one deals with transport mechanisms and roughly speaking with TCP/IP. We will call it the *networking set* and we will work at this level since this is typically the right place where we need to plug IPv6 functionalities.

In order to provide a good integration ability to our package and to make the development of upper-layer objects easier, we consider the structure of the *networking set* and decide to keep the same structure. The first obvious advantage of this choice is code reuse. Not only for the design (and coding) of the package itself but also for network interacting objects. The transition from already existing IPv4 objects, towards an IPv6 networking ability would be simple, flexible and easy to perform. In most cases, it can be automatically done by syntactic transformation (we used `sed`). A second profit is that we did not introduce any new classes. Thus, the underlying semantic of Java is not changed, the security is not depreciated and in a more practical way, network programming with Java remains the same. The last but not the least advantage is the facility of integrating such a package in a JVM. We will see in details in section 4.2 what that really means.

The first decision we had to take in the translation of the networking set from IPv4 to IPv6 was a purely syntactic one. We have to find new names and then upgrade all the references off the concerned objects. This step was performed automatically by using a simple stream editor. In order to handle IPv6 specifications, like the storage size of addresses, the other major modification was focused on the internals of objects. Concept changes between IPv4 and IPv6 were also responsible for some other modifications. Of course, we did not limit ourselves to the already existing interface but improved it by integrating new IPv6 features.

3.2 Implementation of the underlying mechanisms

Under Java, the development of new features, outside the standart Java class library, requires to deal with mechanisms that are themselves not available from the Java$^{\text{TM}}$ Virtual Machine. In our case, these mechanisms deal with operations on IPv6 sockets and addresses. In order to be able to perform such operations, we had to develop a library which would embed the Java$^{\text{TM}}$ virtual machine into native network operations.

In order to build our library we use the *Java*$^{\text{TM}}$ *Native Interface* (JNI). JNI allows Java code that runs inside a JVM to interoperate with applications and libraries written in other programming languages, such as C, C++, and assembly. The main problem we encountered was the question of thread safety. Working in Java induces working in a multi-thread environment, and such an environment requires to take some precautions. Without being *Java-thread safe* the library could not be effective. This is the reason why we could not use classical system calls in the library without any particular precautions.

In fact, only a limited number of system calls were concerned. In particular, they are mostly I/O calls like *open*, *read* or *write*. The important point is that all these calls are already used in the JVM, at least to support the *java.net* package. Thus, the *thread safety* problem has been already addressed. In the JVM, the controversial system calls are mapped in what we will call pseudo functions which take all precautions in connection with threads. Since this pseudo functions were already implemented for the system calls in question, we decided to integrate our library into the architecture of the JVM by proceding the same way as the classical network library. Our C functions would call the same pseudo functions. For more information about network programming, please refer to [19].

4 Results and extensions

The last step in the development of the package was the introduction of specific features provided by IPv6.

4.1 A raw level and new options

In a classical Java environment, raw sockets are not accessible. This could represent a lack of flexibility in several situations. For example, one could want to use *emulators* to reproduce particular environments for testing purposes and thus generate specific traffic (ICMP, router alert...). We could also need to monitor network traffic when using or optimizing parallel applications [2]. For these purposes, we wrote a raw socket interface for Java.

New IPv6 options were also added to the network API. If multicast already existed in Java, IPsec had never been included before in a Java package, though it was defined under IPv4. Flow control is another new feature proposed in the package. It relies on the mechanisms described in section 2.3.

4.2 An IPv6 compliant JVM

The good behavior of our IPv6 package opens new perspectives. Instead of just adding a new package to the JVM, we decided to completely replace the IPv4 network mechanisms present inside a classical JVM by the IPv6 one described above, in order to have a transparent IPv6 compliant JVM. To achieve this task, we replaced the original *java.net* objects with ours and integrated the IPv6 library in the internals of the JVM. The consequence is that any Java objects can not only communicate under IPv6 without any modifications, but also manage IPv4 messages. Indeed, IPv6 provides simple and flexible transition from IPv4. The key transition objective is to allow IPv6 and IPv4 hosts to interoperate. The Simple Internet Transition (SIT) specify for example a model of deployment where all hosts and routers upgraded to IPv6 in the early transition phase are ”dual” capable (*i.e.*, implement complete IPv4 and IPv6 protocol stacks) In other words, SIT ensures that IPv6 hosts can interoperate with IPv4 hosts anywhere in the Internet. This means that our JVM allows any Java program to work under IPv6 as well as under IPv4. In particular, any Java application using networking will work in a IPv6 environment by using indistinctly IPv4 addresses or IPv6 addresses and becomes able to performed *Remote Method Invocations* (RMI) on a IPv4 JVM or in a fully IPv6 network environment. Another interesting point is that the modifications made to the JVM are not irreversible. Indeed, it is very easy to switch between an IPv4/IPv6 JVM and classical IPv4 JVM.

4.3 Results

To validate the IPv6 package, we translated several Java applications from IPv4 to IPv6. This was done by doing only syntactic transformations. We did it for small applications (secured telnet clients, ping programs) and for larger ones (chat programs based on multicast). We used for example `jmrc` (a Java multicast chat program) and introduced flow control and security to it.

To validate the IPv6 compliant JVM, we run several unmodified Java applications on an dual stack platform. The experiments show that both protocols are fully supported. This means, for example, that telnet servers are now accepting IPv6 and IPv4 connections, or that a web server like `Jigsaw` can run without any modification over IPv6. Tests of RMI were also performed and were conclusive since RMI clients and servers now support IPv6 networks.

We are also carried out several experiments and benchmarks, especially concerning the security part and more generally the overhead introduced by IPv6. Unfortunately, at the time of sending the camera ready version of this paper and by a lack of space we can not include graphs but the work will be done soon and we will be able to present our finding in the Workshop in May 2000. First results show that on a local network, IPv4 performs a little better. These results can be easily explained since their is no routers by definition on a local network, and IPv6 was designed to optimize and reduce the latency in each intermediate router.

Our IPv6 API has been developed under several IPv6 stacks for several OS: Linux, FreeBSD, NetBSD and Windows NT. Since the progress of the different stacks are not the same, all the features proposed by the IPv6 package are not always supported in all OS. A new version of the library is under development for Solaris. Note that our package is running under JdK 1.1 and JdK 1.2.

5 Conclusion

We have discussed the design and development of a IPv6 Compliant Java^{TM} Virtual Machine. We have also highlighted the benefits of using IPv6 in distributed applications compare to the IPv4 currently available. Our results have shown that the IPv6 package fulfills the needs of programmers not only in terms of compatibility with IPv4 but also in terms of new functionalities and we hope in terms of performances. The main point is that every Java applications running under our IPv6 JVM are still compatible with IPv4 and thus modification to the application code is not necessary. In order to show that a IPv6 Compliant Java^{TM} Virtual Machine, which is simply a JVM where the networking part has been replaced by our IPv6 package, makes transition from IPv4 to IPv6 free of development/coding, we installed and run the original `Jigsaw` web server. This server `ipv6.loria.fr` can be reached from any hosts IPv4 or IPv6 anywhere on the Internet.

In the future, we plan to continue to improve the development of our IPv6 Compliant Java^{TM} Virtual Machine on other IPv6 stacks. We plan also to develop computational distributed application, especially under the Globus metacomputing systems. For this purpose it seems interesting to take advantage of all the new features provided by IPv6: security, flow control and multicast and we plan to develop an IPv6 version over Myrinet in order to be able to send jumbogram data packets which will provide efficient API for developing MPI in Java [1,7].

References

1. M. Baker, B. Carpenter, G. Fox, and S. H. Koo. mpiJava: An object-oriented Java interface to MPI. In *Parallel and Distributed Processing*, volume 1586 of *Lecture Notes in Computer Science*, pages 748–762, San Juan, Puerto Rico, April 1999.
2. A. M. Bakić, M. W. Mutka, and D. T. Rover. An on-line performance visualization technology. In *Heterogeneous Computing Workshop (HCW '99)*, pages 47–59, San Juan, Puerto Rico, April 1999. IEEE.
3. B. Carpenter, Y.-J. Chang, G. Fox, and X. Li. Java as a language for scientific parallel programming. *Lecture Notes in Computer Science*, 1366, 1998.
4. B. Carpenter, G. Zhang, G. Fox, and X. Li. Towards a Java environment for SPMD programming. *Lecture Notes in Computer Science*, 1470, 1998.

5. C. Chassot, A. Loze, F. Garcia, L. Dairaine, and L. R. Cardenas. Specification and realization of the QoS required by a distributed interactive simulation application in a new generation internet. In M. Diaz, P. Owezarski, and P. Sénac, editors, *Interactive Distributed Multimedia Systems and Telecommunication Services (IDMS'99)*, volume 1718 of *Lecture Notes in Computer Science*, pages 75–91, Toulouse, France, October 1999.
6. G. Cizault. *IPv6: Théorie et pratique*. O'Reilly, 1998.
7. K. Dincer. A ubiquitous message passing interface implementation in java: *jmpi*. In *International Parallel Processing Symposium, Symposium on Parallel and Distributed Processing (IPPS / SPDP 1999)*, pages 203–207, San Juan, Puerto Rico, April 1999. IEEE.
8. Java API documentation. home page. http://java.sun.com/docs/index.html.
9. Robert Fink. Network integration — boning up on IPv6 — the 6bone global test bed will become the new Internet. *Byte Magazine*, 23(3):96NA–3–96NA–8, March 1998.
10. I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, Summer 1997.
11. I. Foster and C. Kesselman. *The Grid: Blueprint for a new computing infrastructure*. Morgan Kaufmann, 1998.
12. G. Fox. Editorial: Java for high-performance network computing. *Concurrency: Practice and Experience*, 10(11–13):821–824, September 1998. Special Issue: Java for High-performance Network Computing.
13. Java Grande. home page. http://www.javagrande.org.
14. Andrew S. Grimshaw, William A. Wulf, James C. French, Alfred C. Weaver, and Paul F. Reynolds, Jr. Legion: The next logical step toward a nationwide virtual computer. Technical Report CS-94-21, Department of Computer Science, University of Virginia, June 08 1994. Mon, 28 Aug 1995 21:06:39 GMT.
15. C. Huitema. *IPv6: The New Internet Protocol*. Prentice Hall, 1996.
16. R.van Nieuwpoort, J. Maassen, T. Bal, H. E.and Kielmann, and R. Veldema. Wide-area parallel computing in java. In *Proceedings of the ACM Java Grande Conference*, New York, NY, June 1999. ACM Press.
17. David Powell. Group communication. *Communications of the ACM*, 39(4):50–97, April 1996. (special section Group Communication).
18. M. J Skidmore, M. J. Sottile, and A. D. Cuny, J. E.and Malony. A prototype notebook-based environment for computational tools. In *High Performance Networking And Computing Conference*, Orlando, USA, November 1998.
19. Richard W Stevens. *UNIX Network Programming*. Software Series. Prentice Hall PTR, 1990.
20. Globus Metacomputing Toolkit. home page. http://www.globus.org.