IP-version Dependency in Application Software

-

Preparing source code for IPv6

stipv6

Stichting IPv6 Nederland

© 2011
Saturnusstraat 60
2516 AH The Hague
The Netherlands
T: 070 - 711 3735
E: info@stipv6.nl

Management Summary

The IT world is reaching a stage where two versions of the Internet Protocol (IP), version 4 and version 6, are active in the IT network environment. As equipment is increasingly being connected using IPv6, a vital question is: is the application software running on this equipment ready to cope with this transition towards IPv6?

Information systems that contain hard-coded IP-version dependencies could be at risk of malfunction if they are exposed to a surrounding network environment that has been made for IPv6[1].

The greatest risk is that applications use knowledge of a particular version of IP, for example, the address format or other properties of the protocol, which does not exist in the same way in the other version. This usage can be programmed directly into the code or indirectly through a dependence upon library calls. This may cause the application to behave incorrectly or may even halt its basic functionality.

Organisations that depend on the correct use of information systems should consider assessing their applications for IP-version dependencies in order to avoid the risk of malfunctioning. They must also realise that hard-coded assumptions about IPv4 represent an even bigger risk.

[1] *See www.sig.eu/en/IPv4-dependency*

Table of contents

1 Introduction

1.1 Internet Protocol and Software

The Internet Protocol (IP) is often associated with routers, switches, computer platforms and networks built on these components. However, modern information systems rely on information supplied and processed by software on computers anywhere that are connected to the Internet.

The IT infrastructure, comprising network components, computer platforms, software libraries and application programs, must ensure that information is processed correctly. A transition of the IT infrastructure to a new version of the Internet Protocol requires software components to operate in a dual-stack environment, which means that both IPv4 and IPv6 protocol stacks are active. Figure 1 gives an abstract overview of the relationship between application, operating system and network infrastructure.

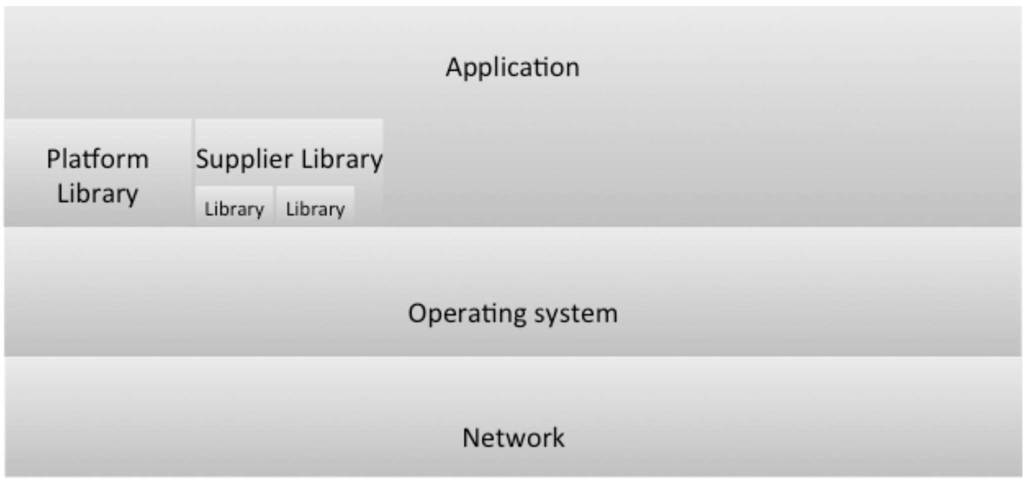

Figure 1 Abstract view of the IT infrastructure

1.2 Purpose and content of this document

The purpose of this document is to describe IP aspects in software, so that these descriptions can be used to help search for, identify and solve any risks that the software may represent with regard to the use of IP. The target reader of this document is the IT manager. Some technical details are presented in the appendices. This document is not intended to be fully exhaustive but rather to serve as a good starting point.

The software dealt with in this document is *application* software rather than systems software. Systems software that contains IP features is supposed to be part of the operating systems, the network components or possibly the libraries. Application software is always dependent on the correct functioning of the systems software it uses, whether it is IP related or not.

2 Categorisation of networking capabilities and software

This chapter describes categories of networking utilisation in software and the types of software that are relevant given the required IP knowledge.

When assessing IP-version dependency, some questions to ask are: does the software system operate in a networked environment? Does the system contain network-dependent features? How is the software system constructed? At which level are networking capabilities being used in the software?

The next sections provide a simple model for what is known as the TCP/IP stack as well as categorisations that should help to identify and localise the risks.

2.1 Simple TCP/IP protocol stack

The Internet Protocol is the binding element in the global IT infrastructure. Applications should use the Internet Protocol determined in the TCP/IP protocol stack. This stack of network communication protocols provides the application with services that enable data transport between applications and users, and between different applications. Application software should use the communication services independently from the underlying protocol layers. Differences in services should be dealt with in the underlying protocol layers. The IP layer is the central layer that binds all services together. This principle is depicted in the figure below.

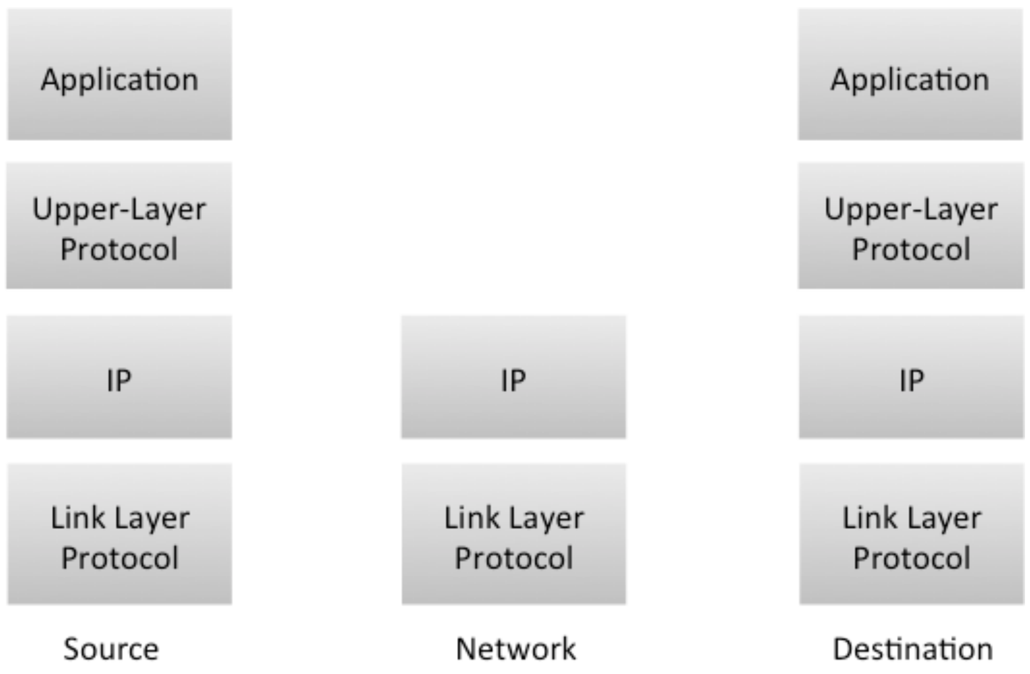

Figure 2 The model exposed by IP to higher-layer protocols and applications[2]

[2] *Figure from IETF journal: Evolution of the IP Model by Dave Thaler http://isoc.org/wp/ietfjournal/?p=454*

2.2 Networking Capabilities

Two categories of IP usage that may contain risks can be identified:

- IP-administration and configuration management: this capability concerns the input of IP address information through a user interface for software-configuration purposes and the output of address information to a log file for logging the maintained connections. The software must be able to cope with both versions of IP.

- Connection management (sockets): this capability relates to initiating and sustaining a connection in order to transport information. IP connections are managed through so-called “sockets”. These are software objects that represent a connection to a destination on a possibly remote network and that can be opened, written to, read from and closed. Connections should be made regardless of the version of IP used.

2.3 Software Categorisation

A software system may comprise different parts that range from off-the-shelf products that can be used “as is” to custom software. A categorisation often used is:

- Package or framework: these types of software are supplied with configuration information on how to compose the functionality and establish network connections

- Third-party library: software that provides packaged services, for example, for IP-connection management in or for a networked environment

- Platform library: software bundled with a platform (UNIX, Windows) that provides, for example, packaged IP services in or for a networked environment

- Custom-developed software: software that contains any or all of the networking capabilities and/or uses the services offered by libraries or frameworks

3 Classification of networking capability risks within software categories

This chapter describes the risks associated with networking capabilities within the software categories identified. It concludes with a few suggestions for mitigating these risks.

3.1 Risks with regard to networking capabilities

Software that is supplied and deployed as a package is ready for use. Such software ought to provide the necessary network services without any burden for the user. The customer should be aware that he or she is dependent on the supplier, and should verify (by questioning and testing) the proper functioning of the package.

Software that provides specific networking capabilities and that is supplied and embedded in applications as a third-party library, requires the correct usage of the API, administration and testing of the services. The supplier of the third-party library is responsible for the proper functioning of the library as a product; the user is responsible for the proper usage of the library by integrating it correctly. If the client is using an open-source library, the client must make sure to apply an up-to-date version of it.

Software that provides specific networking capabilities that are part of a computer platform requires the correct usage of the API as well as the administration and testing of the services. The questions that need to be answered in this respect are: are the services being implemented according to the standards, and do they provide the right communication channels to other platforms? The supplier of the platform can be held responsible for the proper functioning, but the customer is responsible for the proper usage of it by integrating it correctly.

Custom-developed software that provides networking capabilities requires expert domain knowledge for its construction and testing. Only an expert can be expected to program IP networking and its administration correctly. General application developers often do not have these skills and may thus increase the risk of failure. [3]

The table below (table 1) was devised in answer to the question: how do we classify (rate) the IP usage and associated risks, given the software categories? In this table we have classified the occurrence of risks as follows:

+ : IP dependency and risks must be validated but are expected to be small.
0 : IP dependency and risks must be validated and tested, and may be present.
- : IP dependency and risks must be validated and tested; domain knowledge is required.

[3] *An example of this is the usage of broadcasting*

	Administration & Configuration Management	Connection Management
Package / Framework	+	+
Third-party library	0	+
Platform library	-	0
Custom-developed	-	-

Table 1 Occurrence of risks

As explained in the introduction, this classification provides a starting point for the search for elements of risk that may be present in software. Organisations using custom developed software should be aware of these potential risks in their software. They are advised to investigate their software and to mitigate the risks.

3.2 Epilogue

In general, application software should not contain any specific references to the communication protocols that are used. The usage of libraries or other abstraction mechanisms should hide such details. If dependencies on IPv4 do exist, the application may fail when the underlying deployment environment makes the transition from IPv4 to IPv6. Software that is well constructed applies a layered model as illustrated in Figure 2 and does not contain specific knowledge or usage of either IPv4 or IPv6.

Application software that contains hard-coded IP addresses or DNS names is at risk. To mitigate these risks an application programmer should host this information in configuration files. If the programmer must refer to a domain, a host name must be used. This is also the case if a reference to a host address is needed.

Finally, an organisation with application software should not only consider the software item itself but also any software, configuration scripts and log file parsers relating to the single item. The complete chain of software items could be affected by the transition and should therefore be adapted.

A. Appendix A Description of Risks

This section describes the risks in more detail, what exactly they are, how they can be localised and how they can be mitigated.

A.1 IP administration and configuration management

A.1.1 What is IP administration and configuration management?

IP administration and configuration management concerns the input, registering and logging of IP data. Many organisations have custom developed software with this functionality, but the IT department does not maintain this.

A.1.2 What is the related risk?

Data in a specific IP-dependent format is not expected and is thus misinterpreted. The administration software fails unexpectedly.

A.1.3 How to find the elements at risk?

IP address related programming code can be recognised in at least four ways:

- When the IP address is already available in textual form it can be converted to binary form and vice-versa using the following functions:
    - IPv4-only: inet_aton(), inet_ntoa(), inet_addr()
      These functions are deprecated because they do not handle IPv6 addresses. It is therefore not recommended that you use them when developing new software or maintaining existing software
    - IPv4 and IPv6: inet_ntop(), inet_pton()
      These functions handle IP-address conversions for both IPv4 and IPv6 and are recommended for use
- Usage of the name-to-address functions that convert a fully qualified domain name or IP address literal character string into a binary form of an IP address or vice-versa:
    - IPv4-only: gethostbyname(), gethostbyaddr()
      These functions don't work well with IPv6 and can be considered IPv4 only
    - IPv4 and IPv6: getaddrinfo(), getnameinfo()
      These functions supersede the older functions and support IPv4 and IPv6 environments
- Usage of the length of the buffer reserved for an IP address in a character string:
    - IPv4: 15 is the maximum length of a character string representing an IP address
    - IPv6: 39 is the maximum length without IPv4 address embedded
    - IPv6: 45 is the maximum length with IPv4 address embedded
- Common regular expressions that deal with either IPv4 or IPv6 notation and do not handle the other format (See A.1.4)

A.1.4 Examples of regular expressions

In software engineering practice several regular expressions are used to parse or validate IP addresses in character strings. Most of them assume and verify that the address is an

IPv4 address. The following regular expressions were found in a tutorial on the Internet[4], and we assume that they are used literally in source code. The examples only match IPv4 addresses and do not cover IPv6 addresses:

- (\d{1,3}\.){3}\d{1,3}
- \d+\.\d+\.\d+\.\d+
- \d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}
- \[0-9]{1,3}\.\[0-9]{1,3}\.\[0-9]{1,3}\.\[0-9]{1,3}
- ([1-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])(\.([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])){3}
- (25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)
- ([01]?\d\d?|2[0-4]\\d|25[0-5])\.([01]?\d\d?|2[0-4]\d|25[0-5])\.([01]?\d\d?|2[0-4]\d|25[0-5])\.([01]?\d\d?|2[0-4]\d|25[0-5])
- (25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)

In implementations, these expressions might be surrounded by delimiters, white space or regular expression notations such as ^, $ or /b. Depending on the way they are used, they may also contain one or more pairs of capturing parentheses.

Notice that regular expressions that are not literally the same can still be semantically the same. Take this into account when searching for issues related to regular expressions. A possible strategy could be to reverse the test and try to parse IPv4 addresses with the regular expressions in the source code and then examine the expressions that accepted the IPv4 address as valid input.

A.2 Connection Management (Socket handling)

A.2.1 What is a socket?

A socket is one end of an inter-process communication channel. The two processes (client and server) each establish their own sockets. Socket handling concerns the setup, use and finalisation of connections and data communication through these connections.

A.2.2 Goal

Develop application programs that are also IPv6 aware or IPv6 enabled so that they can operate in both IP environments. Programs should work in an environment that supports both IPv4 and IPv6. A specific program may be:

- IPv6-unaware: the program uses deprecated IP library functions and is not able to communicate to IPv6-only destinations
- IPv6-aware: the program is able to communicate with nodes that have IPv6 addresses
- IPv6-featured: the program takes advantage of specific IPv6 features (e.g. flow labels)
- IPv6-required: the program requires IPv6 features and cannot operate in an IPv4 only environment

[4] www.regular-expressions.info

A.2.3 Which risk?

Developed programs may fail in two ways:

- The program is IPv6 unaware: the program runs in an IPv4 only environment and cannot handle IPv6 connections
- The program is constructed for dual stack (both IPv4 and IPv6); the program runs, but the treatment of IP addresses may fail because the programmer lacks knowledge or context. (Note: this risk is not the same as processing IP address data: it is an aspect of the protocol)

A.2.4 How to find IP-version dependencies?

The approach to finding IP-version dependencies is to look for the calls, constants and structures from the specific libraries. In short, this means finding out which libraries are used. Libraries are supplied with platforms (Windows, Unix) and called from specific technologies (C, C++, C# and Java). All technologies share the same risk but may use different libraries and APIs.

The dominant library is the TCP/IP library from 4.3 BSD Unix. This library is ported to other non-UNIX platforms, e.g. Windows (Winsock). The important description here is the RFC 3493 document, which describes the changes to the sockets that support the usage of IPv6. The major changes in the socket API for IPv6 are:

- Possible usage of the protocol family identifier PF_INET6 and address family identifier AF_INET6 for IPv6 only usage, next to PF_INET and AF_INET for IPv4 only usage.
- New socket address structure to carry the IPv6 address.
- New name-to-address and address conversion functions and several new socket options.

	IPv4 specific	**Handling both IPv4 and IPv6**	**IPv6 specific**
Data structures	AF_INET	AF_UNSPEC	AF_INET6
	in_addr sockaddr_in	sockaddr_storage	in6_addr sockaddr_in6

Name-to-address functions	inet_aton()
	inet_addr()
		inet_pton() *	

	inet_ntoa()	inet_ntop() *	

Address conversions functions	gethostbyname()
	gethostbyaddr()
		getnameinfo() *
		getaddrinfo() *	

Table 2 Overview of socket programming[5]

The use of AF_INET or AF_INET6 is only recommended if the programmer specifically wants to use a certain IP protocol address (IPv4 or IPv6). The AF_INET address family

[5] *IPv6 Socket Programming Joonbok Lee KAIST; table adapted to show version independent data structure and functions*

identifier may have been used in the past to exclude non-IP addresses, but will now also exclude IPv6 addresses.

The IPv4 only functions (the blue column in table 2) are not recommended; they are deprecated because they do not handle IPv6 addresses. The data structures are specific for binary representations of IPv4 or IPv6 addresses and are automatically returned in the proper form by the IPv4/IPv6 functions in the middle (green) column.

B. Appendix B Technology details

This section provides details relating to the usage of specific technologies.

B.1 C - UNIX

The dominant language for communication programming is C, and the dominant library is the TCP/IP library from 4.3 BSD Unix. IPv4 dependency in C and C++ can be found by looking for the above constructs and functions. Rino Nucara of GARR[6] has written an introduction about IPv6 programming and IPv4 – IPv6 interoperability.

B.2 MS Windows

Before the .NET era Windows applications were built using the Winsock library, derived from the TCP/IP library. A good starting point for IP knowledge can be found in the IPv6 Guide for Windows Sockets Applications[7].

B.3 .NET

Detailed information on IPv6 network programming for .NET framework 4 can be found in Microsoft's MSDN library[8]. A wealth of information on IPv6 can also be found on Technet[9].

B.4 Java

With the release of J2SE 1.4 in February 2002, Java began supporting IPv6 on Solaris and Linux. Support for IPv6 on Windows was added with J2SE 1.5. While other languages, such as C and C++, can support IPv6, Java has some major advantages:

- With Java you invest in a single code base that is both IPv4 and IPv6 ready
- Your existing Java applications are already IPv6 enabled
- Migration to IPv6 is easy

Java technology abstracts from the specific behaviour and delivers an advantage in this respect.

[6] *See http://www.euchinagrid.org/IPv6/IPv6_presentation/Introduction_to_IPv6_programming.pdf*

[7] *See http://msdn.microsoft.com/en-us/library/ms737579.aspx*

[8] *See http://msdn.microsoft.com/en-us/library/3x7ak53z.aspx*

[9] *See http://technet.microsoft.com/en-us/network/bb530961.aspx*