Foreword

The Java platform and language were conceived with networking support as a core design principle. A measure of its success in this area is how unusual it is today to find a Java application that does not have at least some measure of network awareness or dependence. Developers today routinely build applications and services that a decade ago would have been regarded as highly complex and requiring rare expertise.

Frameworks, containers, and the high-level Java networking APIs have encapsulated this complexity, insulating developers from dealing with many traditional networking issues. However, many developers still make the funamental error of taking this relative simplicity for granted by assuming that interacting across a network is no more complex than interaction with local objects. Many of the poorly performing or scaling applications I have seen are due to naïve decisions taken without considering the ramifications of distribution across a network and without attention to fundamental elements of network programming or configuration.

I was an early reviewer of this book and I admire its economical and thorough but eminently readable style, lucidly describing complex issues without ever outstaying its welcome. This book combines academic rigour with a practical approach deeply informed by real-world experience and I have no hesitation in recommending it to developers of all experience levels. Experienced engineers building network-centric infrastructure or services should not be without this book. In fact, any Java developer building distributed applications such as J2EE, Jini, and Web Services should read this book—at least to understand the fundamental implications of networking on application design and implementation.

Michael Geisler, Sun Microsystems

Preface

THIS BOOK IS INTENDED TO FILL a long-standing gap in the documentation and literature of the Java™ programming language and platform, by providing fundamental and in-depth coverage of TCP/IP and UDP networking from the point of view of the Java API, and by discussing advanced networking programming techniques.[1] The new I/O and networking features introduced in JDK 1.4 provide further justification for the appearance of this text. Much of the information in this book is either absent from or incorrectly specified in the Java documentation and books by other hands, as I have noted throughout.

In writing this book, I have drawn on nearly twenty years' experience in network programming, over a great variety of protocols, APIs, and languages, on a number of platforms (many now extinct), and on networks ranging in size from an Ethernet a few inches in length, to a corporate WAN between cities thousands of miles apart, to the immense geographic spread of the Internet.

This book covers both 'traditional' Java stream-based I/O and so-called 'new I/O' based on buffers and channels, supporting non-blocking I/O and multiplexing, for both 'plain' and secure sockets, specfically including non-blocking TLS/SSL and GSS-API.

Server and client architectures, using both blocking and non-blocking I/O schemes, are discussed and analysed from the point of view of scalability and with a particular emphasis on performance analysis.

An extensive list of TCP/IP platform dependencies, not documented in Java, is provided, along with a handy reference to the various states a TCP/IP port can assume.

1. Sun, Java, and many Java-related terms and acronyms are trademarks of Sun Microsystems Incorporated, Santa Clara, California. These and all other trademarks referred to in this book remain property of their respective owners.

ABOUT THE BOOK

Audience

I have assumed a competent reader familiar with the fundamentals of the Java programming language, specifically the concepts of *class, object, interface, method, parameter, argument, result,* and *exception;* with the basic principles of object-oriented programming: *inheritance* and *polymorphism*; and with the standard Java I/O, utility, and exception classes.

I have also assumed a reader who is able to digest short passages of simple Java code without requiring every line explained, and to turn English prose into Java code without requiring a code sample at every turn. A very basic knowledge of TCP programming with clients and servers is assumed, although I have provided a brief review. Finally, I assume that the reader either knows about the Internet, hosts, and routers, or has the initiative and the resources to look them up.

I have used some of the more standardized vocabulary of design patterns, as first seen in Gamma *et al., Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995, specifically the terms *adapter*, *delegate*, *facade*, and *factory*, which are now in such common use as not to require explanation. I have also used UML sequence diagrams without definition or comment, as these are fairly self-explanatory.

Scope

The book covers TCP downwards from the Java API, through socket options and buffers, to the TCP segment level, including the connection and termination sequences, RST segments, and—to a small extent—windowing, but excluding sequence numbering, pacing, acknowledgements, and retries.

Similarly, it covers UDP downwards from the Java API, through socket options and buffers, to the UDP datagram level, including unicast, broadcast, and multicast, and including material on reliable UDP, erasure codes, and higher-level multicasting protocols.

I have paid particular attention to the neglected and widely misunderstood topic of multi-homed hosts, particularly in relation to UDP unicast, broadcast, and multicast, where multi-homing presents special difficulties.

The TCP, UDP, and TLS/SSL protocols are all covered both in blocking and non-blocking mode, via both traditional Java streams and channel- and buffer-oriented 'NIO' (new I/O). Secure sockets via SSL and TLS are covered in detail, and the JGSS-API is discussed as an alternative.

I have devoted an entire chapter to a reduction-to-practice of the JDK 1.5 SSLEngine, with sample code for a complete and correct SSLEngineManager, making this bizarre apparition actually useable for writing non-blocking SSL servers and clients.

The organization of the book is described in section 1.2.

Exclusions

The book excludes IP at the packet level altogether, as well as associated protocols such as ICMP, ARP, RARP, DHCP, *etc.*, although IGMP does appear fleetingly in the discussion of multicasting. These topics are definitively covered in Stevens & Wright, *TCP/IP Illustrated*, Volumes I and II, Addison-Wesley, 1994–5, whose completeness and authoritativeness I have not attempted to duplicate.

I have deliberately omitted any mention of the defunct 7-layer OSI Reference Model,[2] into which TCP/IP cannot be shoehorned.

I have excluded all higher-level protocols such as HTTP, HTTPS, and FTP. I have also excluded J2EE in its entirety, as well as Java RMI (Remote Method Invocation), with the exception of RMI socket factories which present special, undocumented difficulties. Kathleen McNiff and I have described Java RMI in detail in our book *java.rmi*: *The Guide to Remote Method Invocation*, Addison-Wesley 2001.[3]

I have resisted without apology the recent tendency to re-present all of computer science as design patterns, even in Chapter 12, 'Server and client models', for which design patterns do exist. The relevant parts of Java and the Java Class Library themselves constitute design patterns which subsume many existing patterns for network programming.

This book is about networking, and so is the sample code. Java program code which is not directly relevant to network programming does not appear. Not a line of AWT or Swing code is to be found in these pages, nor are screen shots, console outputs, or examples of streaming audio-visuals or 3D animations. Nor have I presented the 'complete source code' for some arbitrary application of limited relevance.

ACKNOWLEDGEMENTS

I am primarily indebted to the many people who researched and developed the TCP/IP protocol suite over several decades, and whose names appear in the various IETF formal standards and RFCs which define the suite: some of these are listed in the bibliography.

Any serious writer on TCP and UDP owes practically everything to the late W. Richard Stevens, with whom I was privileged to exchange a few e-mails. Stevens documented the entire protocol suite, both the specification and the BSD 4.4 implementation, in his *TCP/IP Illustrated*, 3 volumes, and described the Berkeley Sockets API in all its gruesome details in his *Unix Network Programming*, 2 volumes. These are now fundamental references for anyone who really wants to understand IP network programming in any language.

2. for which see e.g. Piscitello & Chapin, *Open Systems Networking: OSI & TCP/IP*.
3. Much of the present chapter on firewalls first appeared there, and is used by permission.

This book started life in 1993 as a 25-page paper written in collaboration with my brother and colleague David Pitt: the paper was privately distributed to employees and clients, and has subsequently turned up in all sorts of surprising places.

Several anonymous reviewers contributed significantly to the final form and content of this book. All errors however remain mine.

My thanks go to Sun Microsystems Inc. for producing Java and supplying it free of charge, and to Sun Microsystems Ltd, Melbourne, Australia, for providing Solaris and Linux testing facilities.

Thanks also to my long-standing colleague Neil Belford for advice, assistance, and encouragement. Finally, thanks to Tilly Stoové and all the Pitt family for their understanding and support during the writing of this book.

Esmond Pitt, Melbourne, June 2005.

CHAPTER 2

Fundamentals of IP

THIS CHAPTER INTRODUCES the IP protocol and its realization in Java. IP stands for 'Internet protocol', and it is the fundamental protocol of the Internet—the 'glue' which holds the Internet together.

2.1 IP

As RFC 791 says, 'the Internet Protocol is designed for use in interconnected systems of packet-switched computer communication networks'. The Internet is nothing more than a very large number of such systems communicating, via the IP protocol, over various kinds of packet-switched network, including Ethernets and token-rings, telephone lines, and satellite links.

IP is the most fundamental element of a family of protocols collectively known as TCP/IP, consisting of sub-protocols such as ARP—address resolution protocol, RARP—reverse address resolution protocol, ICMP—Internet control message protocol, BOOTP—bootstrap protocol, IGMP—Internet group management protocol, UDP—User datagram protocol, and TCP—Transmission control protocol. This book deals with TCP and UDP; the other protocols mentioned are there to support TCP and UDP in various ways and are not normally the concern of network programmers.

IP consists of *(i)* an addressing system for hosts, *(ii)* the IP packet format definition, and *(iii)* the protocol proper—the rules about transmitting and receiving packets.

IP presently exists in two versions of interest: IPV4, which was the first publicly available version of the protocol, and IPV6, which is in limited use at the time of writing, and which offers a massive expansion of the address space as well as a number of improvements and new features.

2.2 Network addressing

2.2.1 Network interfaces

An Internet host is connected to the network via one or more network interfaces: these are hardware devices, usually manifested as controller cards (network interface controllers or NICs). Each physical network interface may have one or more IP addresses, discussed in the following subsection. In this way, each Internet host has at least one IP address. This topic is discussed further in section 2.3.

2.2.2 IP addresses

An Internet host is identified by a fixed-width 'IP address'. This is a number consists of a 'network' or 'subnet' part, which uniquely identifies the subnetwork within the Internet, and a 'host' part, which uniquely identifies the host within the subnetwork.[1]

In IPv4 an IP address is a 32-bit number, written as a 'dotted-quad' of four 8-bit segments, *e.g.* 192.168.1.24 or 127.0.0.1.

In IPv6 an IP address is a 128-bit number, written as colon-separated quads of 8 bits each, e.g. 0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:1, with the convention that two adjacent colons indicate as many quads of zero as necessary: the address just given can be abbreviated to ::1.

2.2.3 Domain names

The numeric IP addressing system is complemented by an alphabetic naming system known as the Domain Name System or DNS, which partions host names into 'domains' and which provides mappings between IP addresses and host-names, a process known as 'resolution'.

2.2.4 Ports

Each Internet host supports a large number of IP 'ports', which represent individual services within the host, and are identified by a 16-bit 'port number' in the range 1–65535. Many of these port numbers are preallocated: the 'well-known ports' in the range 1–1023, and the 'registered ports' in the range 1024–49151 (0x0400–0xbfff). Servers at the 'well-known ports' require special permission in some operating systems, *e.g.* super-user privilege in Unix-style systems.

1. Readers familiar with NAT—network address translation—will understand that 'uniquely' applies only within the subnet(s) controlled by any single NAT device, but I don't propose to cover NAT in this book.

A specific TCP or UDP service is addressed by the tuple {IP address, port number}. This tuple is also known as a 'socket address'.

2.2.5 *Sockets*

A communications endpoint in a host is represented by an abstraction called a socket. A socket is associated in its local host with an IP address and a port number. In Java, a socket is represented by an instance of one of the java.net classes Socket, ServerSocket, DatagramSocket, or MulticastSocket.

2.2.6 *Network address classes*

In Java, an IP address is represented by a java.net.InetAddress. An IP port is represented in Java by an integer in the range 1–65535, most usually 1024 or above. An IP socket address is represented in Java either by an {IP address, port number} tuple or by the JDK 1.4 SocketAddress class which encapsulates the tuple.

The purposes and uses of the various Java network address classes are explained in Table 2.1.

TABLE 2.1 Network address classes

Name	Description
InetAddress	Represents an IP address or a *resolved* hostname: used for remote addresses. The object cannot be constructed if hostname resolution fails.
InetSocketAddress extends SocketAddress	Represents an IP socket address, *i.e.* a pair {IP address, port} or {hostname, port}. In the latter case an attempt is made to resolve the hostname when constructing the object, but the object is still usable 'in some circumstances like connecting through a proxy' if resolution fails. Can be constructed with just a {port}, in which case the 'wildcard' local IP address is used, meaning 'all local interfaces'.
NetworkInterface	Represents a local network interface, made up of an interface name (e.g. 'leo') and a list of IP addresses associated with the interface. Used for identifying local interfaces in multicasting.

From JDK 1.4, the InetAddress class is abstract and has two derived classes: Inet4Address for IPV4 and Inet6Address for IPV6. You really don't need to be aware of the existence of these derived classes. You can't construct them: you obtain instances of them via static methods of InetAddress, and you are generally better off just assuming that they are instances of InetAddress. The only differ-

ence between the derived classes from the point of view of the programmer is the Inet6Address.isIpV4CompatibleAddress method, which returns true if 'the address is an IPV4 compatible IPV6 address; or false if address is an IPV4 address'.[2] It is a rare Java program which needs to be concerned with this.

2.2.7 *Special* IP *addresses*

In addition to the IP addresses belonging to its network interface(s), an Internet host has two extra IP addresses, which are both usable only within the host, as shown in Table 2.2.

TABLE 2.2 Special IP addresses

Name	IPV4	IPV6	Description
loopback	127.0.0.1	::1	This is used to identify services the local host in situations where the host's external DNS name or IP address are unavailable or uninteresting, *e.g.* in a system which is only intended to communicate within a single host.
wildcard	0.0.0.0	::0	This is used when creating sockets to indicate that they should be bound to 'all local IP addresses' rather than a specific one. This the normal case. In Java it can be indicated by an absent or null InetAddress.

The InetAddress class exports a number of methods which enquire about the attributes of an address. These methods are summarized in Table 2.3.

TABLE 2.3 InetAddress methods

Name	Meaning if 'true'[a]
isAnyLocalAddress	Wildcard address: see Table 2.2.
isLinkLocalAddress	Link-local unicast address. Undefined in IPV4; in IPV6 it is an address beginning with FE:80.
isLoopback	Loopback address: see Table 2.2.
isMCGlobal	Multicast address of global scope.
isMCLinkLocal	Multicast address of link-local scope.

2. JDK 1.4 online documentation.

TABLE 2.3 InetAddress methods

Name	Meaning if 'true'[a]
isMCNodeLocal	Multicast address of node-local scope.
isMCOrgLocal	Multicast address of organization-local scope.
isMCSiteLocal	Multicast address of site-local scope.
isMulticastAddress	Multicast address. In IPV4 this is an address in the range 224.0.0.0 to 239.255.255.255; in IPV6 it is an address beginning with FF.
isSiteLocal	Site-local unicast address. Undefined in IPV4; in IPV6 it is an address beginning with FE:C0.

a. The IPV6 cases refer to the specifications in RFC 2373.

The methods isMCGlobal, isMCLinkLocal, etc which return information about multicast address scopes are discussed in section 11.1.4.

2.3 MULTI-HOMING

A multi-homed host is a host which has more than one IP address. Such hosts are commonly located at gateways between IP subnets, and commonly have more than one physical network interface. It is really only in such hosts that programmers need to be concerned with specific local ip addresses and network interfaces.

Network interfaces were practically invisible in Java prior to JDK 1.4, which introduced the NetworkInterface class. From JDK 1.4, the network interfaces for a host can be obtained with the methods:

```
class NetworkInterface
{
  static Enumeration  getNetworkInterfaces()
                        throws SocketException;
  Enumeration         getInetAddresses();
}
```

where getNetworkInterfaces returns an Enumeration of NetworkInterfaces, and getInetAddresses returns an Enumeration of InetAddresses, representing all or possibly a subset of the IP addresses bound to a single network interface. If there is no security manager, the list is complete; otherwise, any InetAddress to which access is denied by the security manager's checkConnect method is omitted from the list.

The accessible IP addresses supported by a host can therefore be retrieved by the code sequence of Example 2.1.

```
// Enumerate network interfaces (JDK >= 1.4)

Enumeration    interfaces
   = NetworkInterface.getNetworkInterfaces();
while (interfaces.hasMoreElements())
{
   NetworkInterface        intf
      = (NetworkInterface)nwifs.nextElement();

   // Enumerate InetAddresses of this network interface
   Enumerationaddresses = intf.getInetAddresses();
   while (addresses.hasMoreElements())
   {
     InetAddress  address
       = (InetAddress)addresses.nextElement();
     // …
   }
}
```

EXAMPLE 2.1 Enumerating the local network interfaces

2.4 IPV6

Java has always supported IPV4, the original version of the IP protocol. IPV6 is the next version of IP, which is intended to improve a number of aspects of IPV4 including efficiency; extensibility; the 32-bit IPV4 address space; quality-of-service support; and transport-level authentication and privacy.

From JDK 1.4, Java also supports IPV6 where the host platform does so, and it is completely transparent to the programmer. Your existing Java networking program automatically supports both IPV4 and IPV6 if executed under JDK 1.4 on a platform supporting IPV6: you can connect to both IPV4 and IPV6 servers, and you can be an IPV4 and IPV6 server, accepting connections from both IPV4 and IPV6 clients.

2.4.1 Compatibility

IPV6 supports IPV4 via 'IPV4-compatible addresses'. These are 128-bit IPV6 address whose high-order 96 bits are zero. For example, the IPV4 address 192.168.1.24 can used in IPV6 as the IPV4-compatible address ::192.168.1.24.

Java's IPV6 support can be controlled via system properties. These allow you to disable IPV6 support, so that only IPV4 clients and servers are supported. You cannot disable IPV4 support via these properties, although you can achieve the same effect by specifying only IPV6 network interfaces as local addresses

when creating or binding sockets or server sockets. In future there will be a socket option to obtain IPV6-only behaviour on a per-socket basis.[3]

These system properties are described in Table 2.4.

TABLE 2.4 IPV6 system properties

Name	Values	Description
java.net .preferIPv4Stack	false (default), true	By default, IPV6 native sockets are used if available, allowing applications to communicate with both IPV4 andIPv6 hosts. If this property is set to true, IPV4 native sockets are always used. The application will not be able to communicate with IPV6 hosts.
java.net .preferIPv6Addresses	false (default), true	By default, if IPV6 is available, IPV4-mapped addresses are preferred over IPV6 addresses, 'for backward compatibility—e.g. applications that depend on an IPV4-only service, or ... on the ["dotted-quad"] representation of IPV4 addresses'. If this property is set to true, IPV6 addresses are preferred over IPV4-style addresses, 'allowing applications to be tested and deployed in environments where the application is expected to connect to IPV6 services'.[a]

a. Both quotations from Java 1.4 IPV6 User Guide.

2.4.2 *Programming differences in Java*

In any situation where you need to determine dynamically whether you have an IPV4 or an IPV6 socket, the following technique can be used:

```
if (socket.getLocalAddress() instanceof Inet6Address)
  ; // you have an IPv6 socket
else
  ; // you have an IPv4 socket
```

3. The Java IPV6 User Guide is distributed in the JDK *Guide to Features—Networking*, and is available online at http://java.sun.com/j2se/1.5/docs/guide/net/ipv6_guide/index.html.

Apart from the formats of actual IP addresses, the java.net.Inet6SocketAddress class described in section 2.2 and the Socket.setTrafficClass method described in section 3.19 are the only points in the entire java.net package where you need to be concerned with IPV4 and IPV6.