

# *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.<sup>1</sup>

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 partitions 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
<code>InetAddress</code>	Represents an IP address or a <i>resolved</i> hostname: used for remote addresses. The object cannot be constructed if hostname resolution fails.
<code>InetSocketAddress</code> extends <code>SocketAddress</code>	Represents an IP socket address, <i>i.e.</i> 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’.
<code>NetworkInterface</code>	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’.<sup>2</sup> 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 <code>DNS</code> name or <code>IP</code> address are unavailable or uninteresting, <i>e.g.</i> 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 <code>IP</code> addresses’ rather than a specific one. This the normal case. In Java it can be indicated by an absent or null <code>InetAddress</code> .

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’ <sup>a</sup>
<code>isAnyLocalAddress</code>	Wildcard address: see Table 2.2.
<code>isLinkLocalAddress</code>	Link-local unicast address. Undefined in <code>IPv4</code> ; in <code>IPv6</code> it is an address beginning with <code>FE80</code> .
<code>isLoopback</code>	Loopback address: see Table 2.2.
<code>isMCGlobal</code>	Multicast address of global scope.
<code>isMCLinkLocal</code>	Multicast address of link-local scope.

2. [JDK 1.4 online documentation](#).

TABLE 2.3 InetAddress methods

Name	Meaning if 'true' <sup>a</sup>
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 <code>IPv4</code> this is an address in the range 224.0.0.0 to 239.255.255.255; in <code>IPv6</code> it is an address beginning with <code>FF</code> .
isSiteLocal	Site-local unicast address. Undefined in <code>IPv4</code> ; in <code>IPv6</code> it is an address beginning with <code>FE:C0</code> .

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)interfaces.nextElement();

    // Enumerate InetAddresses of this network interface
    Enumeration addresses = 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 be 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.<sup>3</sup>

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 and IPv6 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’. <sup>a</sup>

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](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.





<http://www.springer.com/978-1-84628-030-6>

Fundamental Networking in Java

Pitt, E.

2006, XVIII, 382 p. 23 illus., Hardcover

ISBN: 978-1-84628-030-6