

Module 3 sample

Handling protocols with the Net component

3.1 Getting to know the protocols.....	1
3.2 The Net API.....	13
3.3 Creating a multiprotocol handler.....	23
3.4 Summary.....	32
Index.....	33

The Net component brings together implementations for a diverse range of Internet protocols. It's a feature-rich component and had been around a long time before it was open-sourced through Jakarta Commons. (It was originally built by ORO, Inc.)

Most programmers have to deal with only a subset of the vast array of Internet protocols. The ones that are used most often include Hypertext Transfer Protocol (HTTP), File Transfer Protocol (FTP), and, perhaps, the mail protocols: Simple Mail Transfer Protocol (SMTP) and Post Office Protocol (POP3). However, several lesser-known protocols are also in use. The Net component features both the well known and the not so well known protocols in an easy-to-use interface.

In this chapter, we'll explore the Net component and look at the protocols it makes available. This will help you understand the basics of these protocols before you begin using them. Next, we'll examine the structure of the Net component and explore its API. Finally, we'll use all this information to develop a multiprotocol handler that uses each of the protocols.

3.1 Getting to know the protocols

A *protocol*, in the real world, is a way something should be done. By adhering to a protocol, you're following a strict procedure for doing certain things—for example, you shouldn't take the last helping of dessert without asking a dinner guest if they'd like to have it!

In the computer world, a protocol is a previously agreed upon way for two machines to exchange information with each other. Without a protocol to guide and define how machines talk to each other, there would be anarchy and confusion. A protocol may determine several things: how the two machines will handshake, how the transmitting machine will initiate transfer of data, what the format of the data will be, what the recipient machine will do to indicate that it has received the data, what the recipient machine will do to indicate an error condition, and so on.

3.1.1 TCP and UDP: the building blocks

Before we talk about the protocols covered by the Net component, let's discuss the protocols that are the building blocks of these protocols. A discussion of network technologies is incomplete without a basic understanding of the TCP and UDP protocols, which are the low-level protocols that act as the message carriers for the high-level, application-specific protocols.

The Transmission Control Protocol (TCP) is specified in rfc793.¹ It's a reliable, connection-oriented, complex protocol:

- It's *reliable* because data sent by TCP can't be lost: The protocol marks all packets of information that it transmits with a sequence number. This allows for retransmission of packets that go missing, because the receiving end can ask for those packets by looking up the sequence numbers of the packets it receives.
- It's *connection-oriented* because a connection must be established between machines before data can be exchanged between them.
- It's *complex* because it requires error correction and retransmission policies built in at the protocol layer itself.

The User Datagram Protocol (UDP) is specified in rfc768. It's a nonreliable, connectionless, simple protocol:

- It's *nonreliable* because packets sent via UDP may or may not reach their destination: The packets may get lost along the route due to incorrect addressing or checksum errors.
- It's *connectionless* because each packet is self-contained with the source host and destination host address. No direct connection stream is established between the source and the destination machines.
- It's *simple* because it doesn't require a connection, and no error checking or retransmission of packets is involved. This also means that if an application-level protocol is using UDP as the underlying protocol for transmission of data, it must be ready to accept loss of data or handle error checking and retransmission of lost data on its own.

It helps to think of TCP as a continuous phone line communication channel and UDP as a standard postal letter communication channel. With TCP, a continuous full duplex (two-way) channel is established before any data is exchanged, similar to numbers being dialed before a phone call begins. With UDP, a letter is marked with a destination address and is posted; it will probably reach its destination, but it may not.

Both TCP and UDP are protocols on the Transport layer of the standard TCP/IP four-layer model, which is shown in figure 3.1. For this reason, these protocols are low-level protocols as compared to, for example, HTTP, which is considered a high level protocol.

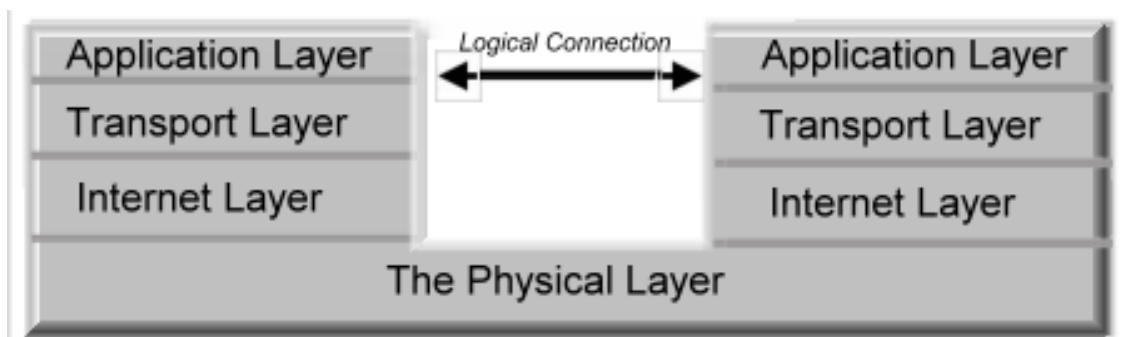


Figure 3.1 The four-layer TCP/IP model, showing the different levels of protocols

3.2.2 High- and low-level protocols

Whenever an application-level protocol is used, it's likely to be running on top of other low-level protocols. Consider HTTP. Although this protocol is designed to transfer information between a web browser and a

¹ See module 2, section 2.1.1, for the definition of an RFC.