4th Edition

Computer Networking

A Top-Down Approach

James F. Kurose

University of Massachusetts, Amherst

•

Keith W. Ross

Polytechnic University, Brooklyn

PEARSON Addison Wesley

Boston San Francisco New York
London Toronto Sydney Tokyo Singapore Madrid
Mexico City Munich Paris Cape Town Hong Kong Montreal

DOCKET ALARM Find authenticated court documents without watermarks at docketalarm.com.

Publisher	Greg Tobin
Executive Editor	Michael Hirsch
Assistant Editor	Lindsey Triebel
Associate Managing Editor	Jeffrey Holcomb
Senior Production Supervisor	Marilyn Lloyd
Senior Marketing Manager	Michelle Brown
Marketing Assistant	Sarah Milmore
Cover Designer	Joyce Cosentino Wells
Cover Image	©Robert Harding/Getty Images: Bridge over Dordogne River, Aquitaine, France
Art Director	Janet Theurer/Theurer Briggs Design
Art Studio	Patrice Rossi Calkin/Rossi Illustration & Design
Senior Manufacturing Buyer	Carol Melville
Senior Media Producer	Bethany Tidd
Project Management	Nancy Kotary, Alicia Williams, Scott Harris/Argosy Publishing, Inc.
Composition	Argosy Publishing, Inc.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and Addison-Wesley was aware of a trademark claim, the designations have been printed in initial caps or all caps.

The programs and applications presented in this book have been included for their instructional value. They have been tested with care, but are not guaranteed for any particular purpose. The publisher does not offer any warranties or representations, nor does it accept any liabilities with respect to the programs or applications.

Library of Congress Cataloging-in-Publication Data
Kurose, James F.
Computer networking : a top-down approach / James F. Kurose, Keith W. Ross. — 4th ed.
p. cm.
ISBN 0-321-49770-8
1. Internet. 2. Computer networks. I. Ross, Keith W., 1956- II. Title.
TK5105.875.I57K88 2007
004.6—dc22 2007002094

Copyright © 2008 Pearson Education, Inc.
All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher. Printed in the United States of America. For information on obtaining permission for use of material in this work, please submit a written request to Pearson Education, Inc., Rights and Contracts Department, 75 Arlington Street, Suite 300, Boston, MA 02116, fax your request to 617-848-7047, or e-mail at http://www.pearsoned.com/ legal/permissions.htm.

ISBN-13: 978-0-321-49770-3
ISBN-10: 0-321-49770-8

3 4 5 6 7 8 9 10—CRW—10 09 08

DOCKET ALARM Find authenticated court documents without watermarks at docketalarm.com.

The Interface Between the Process and the Computer Network

As noted above, most applications consist of pairs of communicating processes, with the two processes in each pair sending messages to each other. Any message sent from one process to another must go through the underlying network. A process sends messages into, and receives messages from, the network through a software interface called a **socket**. Let's consider an analogy to help us understand processes and sockets. A process is analogous to a house and its socket is analogous to its door. When a process wants to send a message to another process on another host, it shoves the message out its door (socket). This sending process assumes that there is a transportation infrastructure on the other side of its door that will transport the message to the door of the destination process. Once the message arrives at the destination host, the message passes through the receiving process's door (socket), and the receiving process then acts on the message.

Figure 2.3 illustrates socket communication between two processes that communicate over the Internet. (Figure 2.3 assumes that the underlying transport protocol used by the processes is the Internet's TCP protocol.) As shown in this figure, a socket is the interface between the application layer and the transport layer within a host. It is also referred to as the **Application Programming Interface (API)** between the application and the network, since the socket is the programming interface with which network applications are built. The application developer has control of everything on the application-layer side of the socket but has little control of the transport-layer side of the socket. The only control that the application developer has on the transport-layer side is (1) the choice of transport protocol and (2) perhaps the ability to fix a few transport-layer parameters such as maximum buffer

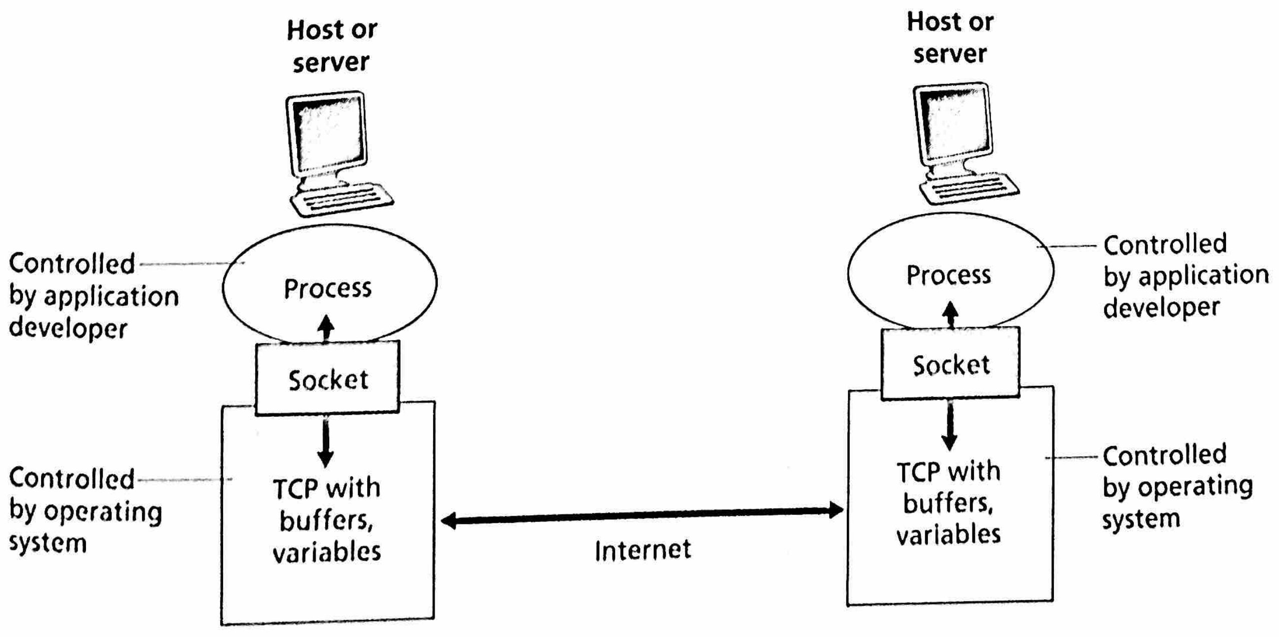
Host or server
Host or server
Controlled by application developer
Process
Socket
Controlled by operating system
TCP with buffers, variables
Internet
Process
Socket
TCP with buffers, variables
Controlled by application developer
Controlled by operating system

Figure 2.3 ♦ Application processes, sockets, and underlying transport protocol

DOCKET ALARM Find authenticated court documents without watermarks at docketalarm.com.

necessarily conform to any existing RFC. A single developer (or development team) creates both the client and server programs, and the developer has complete control over what goes in the code. But because the code does not implement a public-domain protocol, other independent developers will not be able to develop code that interoperates with the application. When developing a proprietary application, the developer must be careful not to use one of the well-known port numbers defined in the RFCs.

In this and the next section, we examine the key issues in developing a proprietary client-server application. During the development phase, one of the first decisions the developer must make is whether the application is to run over TCP or over UDP. Recall that TCP is connection oriented and provides a reliable byte-stream channel through which data flows between two end systems. UDP is connectionless and sends independent packets of data from one end system to the other, without any guarantees about delivery.

In this section we develop a simple client application that runs over TCP; in the next section, we develop a simple client application that runs over UDP. We present these simple TCP and UDP applications in Java. We could have written the code in C or C++, but we opted for Java mostly because the applications are more neatly and cleanly written in Java. With Java there are fewer lines of code, and each line can be explained to the novice programmer without much difficulty. But there is no need to be frightened if you are not familiar with Java. You should be able to follow the code if you have experience programming in another language.

For readers who are interested in client/server programming in C, there are several good references available [Donahoo 2001; Stevens 1997; Frost 1994; Kurose 1996].

2.7.1 Socket Programming with TCP

Recall from Section 2.1 that processes running on different machines communicate with each other by sending messages into sockets. We said that each process was analogous to a house and the process's socket is analogous to a door. As shown in Figure 2.30, the socket is the door between the application process and TCP. The application developer has control of everything on the application-layer side of the socket; however, it has little control of the transport-layer side. (At the very most, the application developer has the ability to fix a few TCP parameters, such as maximum buffer size and maximum segment size.)

Now let's take a closer look at the interaction of the client and server programs. The client has the job of initiating contact with the server. In order for the server to be able to react to the client's initial contact, the server has to be ready. This implies two things. First, the server program cannot be dormant—that is, it must be running as a process before the client attempts to initiate contact. Second, the server program must have some sort of door—more precisely, a socket—that welcomes some initial contact from a client process running on an arbitrary host. Using our house/door analogy for a process/socket, we will sometimes refer to the client's initial contact as "knocking on the welcoming door."

DOCKET ALARM Find authenticated court documents without watermarks at docketalarm.com.

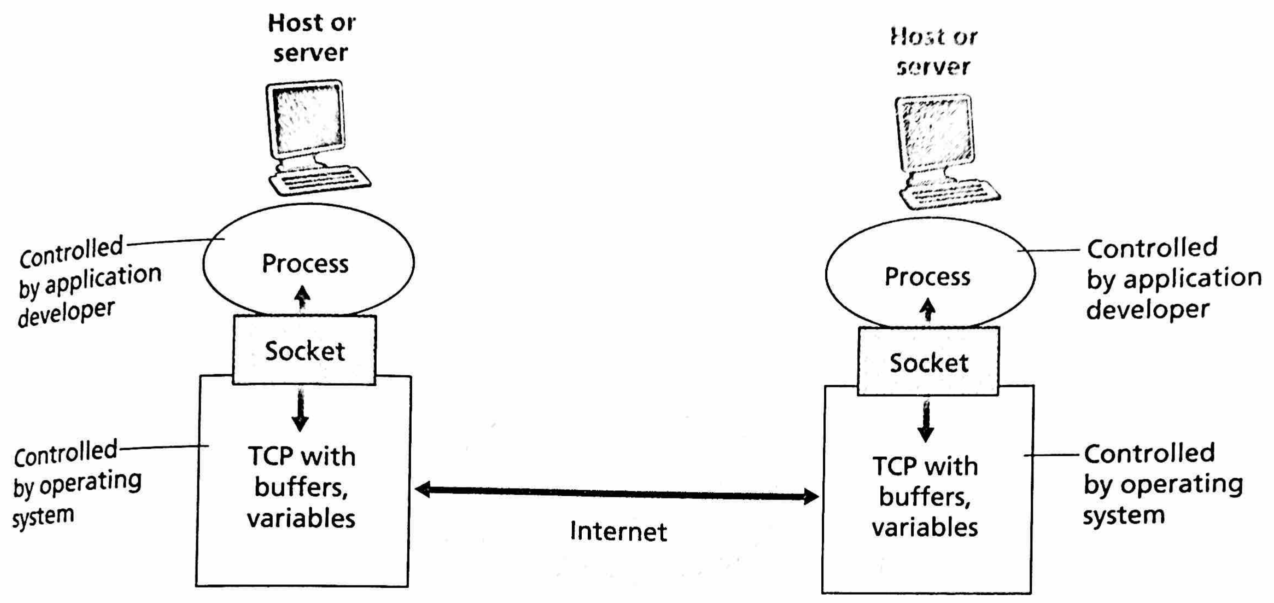
Host or server
Host or server
Controlled by application developer
Process
Socket
Controlled by operating system
TCP with buffers, variables
Internet
Process
Socket
TCP with buffers, variables
Controlled by application developer
Controlled by operating system

Figure 2.30 ♦ Processes communicating through TCP sockets

With the server process running, the client process can initiate a TCP connection to the server. This is done in the client program by creating a socket. When the client creates its socket, it specifies the address of the server process, namely, the IP address of the server host and the port number of the server process. Once the socket has been created in the client program, TCP in the client initiates a three-way handshake and establishes a TCP connection with the server. The three-way handshake, which takes place at the transport layer, is completely transparent to the client and server programs.

During the three-way handshake, the client process knocks on the welcoming door of the server process. When the server "hears" the knocking, it creates a new door—more precisely, a new socket—that is dedicated to that particular client. In our example below, the welcoming door is a `ServerSocket` object that we call the `welcomeSocket`. When a client knocks on this door, the program invokes `welcomeSocket`'s `accept()` method, which creates a new door for the client. At the end of the handshaking phase, a TCP connection exists between the client's socket and the server's new socket. Henceforth, we refer to the server's new, dedicated socket as the server's **connection socket.**

From the application's perspective, the TCP connection is a direct virtual pipe between the client's socket and the server's connection socket. The client process can send arbitrary bytes into its socket, and TCP guarantees that the server process will receive (through the connection socket) each byte in the order sent. TCP thus provides a **reliable byte-stream service** between the client and server processes. Furthermore, just as people can go in and out the same door, the client process not only sends bytes

DOCKET ALARM Find authenticated court documents without watermarks at docketalarm.com.