The Transport Layer: Tutorial and Survey

SAMI IREN and PAUL D. AMER

University of Delaware

AND

PHILLIP T. CONRAD

Temple University

Transport layer protocols provide for end-to-end communication between two or more hosts. This paper presents a tutorial on transport layer concepts and terminology, and a survey of transport layer services and protocols. The transport layer protocol TCP is used as a reference point, and compared and contrasted with nineteen other protocols designed over the past two decades. The service and protocol features of twelve of the most important protocols are summarized in both text and tables.

Categories and Subject Descriptors: C.2.0 [**Computer-Communication Networks**]: General—*Data communications*; *Open System Interconnection Reference Model* (OSI); C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Network communications*; *Packet-switching networks*; *Store and forward networks*; C.2.2 [**Computer-Communication Networks**]: Network Protocols; *Protocol architecture* (OSI model); C.2.5 [**Computer-Communication Networks**]: Local and Wide-Area Networks

General Terms: Networks

Additional Key Words and Phrases: Congestion control, flow control, transport protocol, transport service, TCP/IP

1. INTRODUCTION

In the OSI 7-layer Reference Model, the *transport layer* is the lowest layer that operates on an end-to-end basis between two or more communicating hosts. This layer lies at the boundary between these hosts and an internetwork of routers, bridges, and communication links that moves information between hosts. A good transport layer service (or simply, *transport service*) allows applications to use a standard set of primitives and run on a variety of networks without worrying about different network interfaces and reliabilities.

This work was supported, in part, by the National Science Foundation (NCR 9314056), the U.S. Army Research Office (DAAL04-94-G-0093), and the Adv Telecomm/Info Dist'n Research Program (ATIRP) Consortium sponsored by ARL under Fed Lab Program, Cooperative Agreement DAAL01-96-2-0002.

Authors' addresses: S. Iren and P. D. Amer, Department of Computer and Information Sciences, University of Delaware, Newark, DE 19716; P. T. Conrad, Department of Computer and Information Sciences, Temple University, Philadelphia, PA 19122.

CONTENTS

Essentially, the transport layer isolates applications from the technology, design, and idiosyncracies of the network.

Dozens of transport protocols have been developed or proposed over the last two decades. To put this research in perspective, we focus first on the features of probably the most well-known transport protocol — namely the Internet's *Transmission Control Protocol* (TCP) — and then contrast TCP with many alternative designs.

Section 2 introduces the basic concepts and terminology of the transport layer through a simple example illustrating a TCP connection. Section 3 surveys the range of different *services* that can be provided by a transport layer. Similarly, Section 4 surveys the range of *protocol* designs that provide these services. (The important distinction between service and protocol is a major theme throughout this paper.) Section 5 briefly surveys nine widely implemented transport protocols other than TCP (UDP, TP0, TP4, SNA-APPN, DECnet-NSP, ATM, XTP, T/TCP and RTP) and two others that, although not widely implemented, have been particularly influential (VMTP and NETBLT). This section also includes briefer descriptions of eight experimental protocols that appear in the research literature (Delta-t, MSP, SNR, DTP, k-XP, TRUMP, POC, and TP++). Section 6 concludes the paper with an overview of past, current, and future trends that have influenced transport layer design including the impact of wireless networks. This section also presents a few of the debates concerning transport protocol design. As an appendix, tables are provided summarizing TCP and eleven of the transport protocols discussed in Section 5. Similar tables for the experimental protocols are omitted for reasons of space, but are available on the authors' Web site: `www.eecis.udel.edu/˜amer/PEL/survey/`.

This survey concentrates on *unicast* service and protocols — that is, communication between exactly two hosts (or two host processes). *Multicast* protocols [Armstrong et al. 1992; Bormann et al. 1994; Braudes and Zabele 1993; Deering 1989; Floyd et al. 1995; McCanne et al. 1996; Smith and Koifman 1996] provide communication among $n \geq 2$ hosts. Multicast represents an important research area currently undergoing significant change and development, and is worthy of a separate survey.

A previous study surveying eight transport protocols can be found in Doeringer et al. [1990].

2. TRANSPORT LAYER CONCEPTS AND TERMINOLOGY

From an application programmer's perspective, the transport layer provides interprocess communication between two processes that most often are running on different hosts. This section introduces some basic transport layer concepts and terminology through an example: a simple document retrieval over the World Wide Web (herein Web) utilizing the TCP transport protocol.

2.1 Introduction to TCP

Although we provide a broad survey of the transport layer, the service and protocol features of TCP are used throughout this paper as a point of reference.

Over the last two decades the Internet protocol suite (also called the TCP/IP protocol suite) has come to be the most ubiquitous form of computer networking. Hence, the most widely used transport protocols today are TCP and its companion transport protocol, the User Datagram Protocol (UDP). A few other protocols are widely used, mainly because of their connection to the proprietary protocol suites of particular vendors. Examples include the transport protocols from IBM's SNA, and Digital's DECnet. However, the success of the Internet has led nearly all vendors in the direction of TCP/IP as the future of networking.

The Internet's marked success would not alone be sufficient justification for organizing a survey around a single protocol. Also important is that TCP provides examples of many significant issues that arise in transport protocol design. The design choices made in TCP have been the subject of extensive review, experimentation, and large-scale experience, involving the best researchers and practitioners in the field. TCP represents the culmination of many years of thought about transport protocol design.

A final reason that TCP provides a good starting point for study, is that the history of research and development on TCP can be traced in publicly available documents. Ongoing research and development of transport protocols, particularly TCP, is the focus of two working groups of the Internet Society. The *end2end* working group of the Internet Research Task Force (IRTF, `www.irtf.org`) discusses ongoing long-term research on transport protocols in general (including TCP), while the *tcp-impl* group of the Internet Engineering Task Force (IETF, `www.ietf.org`) focuses on short-term TCP implementation issues. Both groups maintain active mailing lists where ideas are discussed and debated openly. The work of these groups can be found in journal articles, conference proceedings, and documents known as Internet Drafts and Requests for Comments (RFCs). RFCs contain not only all the Internet Standards, but also other information of historical and technical interest. It is much more difficult for researchers to obtain similar information concerning proprietary protocols.

2.2 General Role of the Transport Layer

To illustrate the role that the transport layer plays in a familiar application, the remainder of Section 2 examines the role of TCP in a simple interaction over the Web.

The Web is an example of a client/server application. A human interacts with a Web browser (client) running on a "local" machine. The Web browser communicates with a server on some "remote" machine. The Web uses an application layer protocol called the Hypertext Transfer Protocol (HTTP) [Berners-Lee et al. 1996]. HTTP is a simple request/response protocol. Suppose, for example, that you have a personal computer with Internet access, and you wish to retrieve the page "`http://www.eecis.udel.edu/research.html`" from the University of Delaware Web site. In the simplest case,[1] the client sends a request containing the filename of the desired Web page ("`GET /research.html`") to a server ("`www.eecis.udel.edu`"), and the server sends back a response consisting of the contents of that file.

This communication takes place over a complex internetwork of computers that is constantly changing in terms of both technology and topology. A connection between two particular hosts may involve such diverse technologies as Ethernet, Token Ring, X.25, ATM, PPP, SONET, just to name a few. However, a programmer writing a Web client or server does not want to be concerned with the details of *how* communication takes place between client and server. The programmer simply wants to send and receive messages in a way that does not change as the underlying network changes. This is the function of the transport layer: to provide an abstraction of interprocess communication that is independent of the underlying network.

HTTP uses TCP as the transport layer. The programmer writing code for an HTTP client or server would access TCP's service through function calls that comprise that transport layer's *Application Program Interface* (API). At a minimum, a transport layer API provides functions to send and receive messages; for example, the *Berkeley Sockets* API provides functions called `write()` and `read()` (for more details, see Stevens [1998]).

Because TCP is *connection-oriented*, the Berkeley Sockets API also provides a `connect()` function for setting up a *connection* between the local and remote processes. It also provides a `close()` function for closing a connection. Note that while TCP is connection-oriented, not all transport services establish a connection before data is sent. Connection-oriented and connectionless services and protocols are discussed in Sections 3.1, 3.2.5 and 4.1.

[1]To simplify the discussion, we will assume HTTP version 0.9 and a document containing only hypertext: no inline images, applets, etc. This avoids discussion of HTTP 1.0 headers, persistent connections as in HTTP 1.1, (which complicate the issue of how and when the connection is closed,) and the necessity for multiple connections where inline images are involved.

2.3 Terminology: SDUs, PDUs, and the like

One difficulty in summarizing any topic is the wide range of terms used for similar concepts. Throughout this paper, we use a simplified communication model (Figure 1) that employs some OSI terminology. At the top layer, a *user sender* (e.g., a Web client) has some messages to communicate to the *user receiver* (e.g., a Web server). These so-called *application entities* use the service of the transport layer. Communication between *peer* entities consists of an exchange of *Protocol Data Units* (PDUs). Application peers communicate using Application PDUs (APDUs), while transport peers communicate using Transport PDUs (TPDUs), etc. In our Web example, the first APDU is the request "`GET /research.html`" sent from the client (application entity) to the server (its peer application entity). The Web server will respond with an APDU containing the entire text of the file "`research.html`".

Many transport and application protocols are *bidirectional*; that is, both sides can send and receive data simultaneously. However, it is frequently useful to focus on one direction while remaining aware that the other direction is also operational. As Figure 1 shows, each application entity can assume both the role of sender and receiver; for the APDU "`GET /research.html`", the client is the user sender and the server is the user receiver (as shown by more prominent labels). When the APDU containing the contents of the file "`research.html`" is sent, user sender and user receiver reverse roles (as indicated by the dotted line boxes, and the lighter italicized labels).

The term *transport entity* refers to the

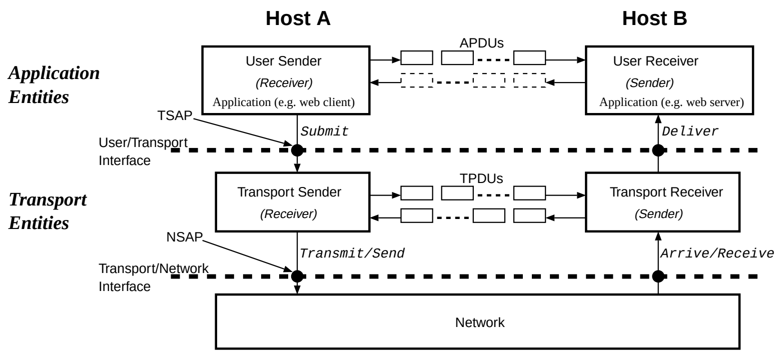

Figure 1. Transport service.

hardware and/or software within a given host that implements a particular transport service and protocol. Again, even when the protocol is bidirectional, we focus on one direction for purposes of clarity. In this model, the user sender *submits* a chunk of user data (i.e., a *Transport Service Data Unit* (TSDU), or informally, a *message*) to the *transport sender*. The transport sender *transmits* or *sends* this data to the *transport receiver* over a *network* which may provide different levels of reliability. The transport receiver *receives* the data that *arrives* from the network and *delivers* it to the user receiver. Note that even when one transport entity assumes the role of sender and the other assumes the role of receiver, we use solid lines to show TPDUs flowing in both directions. This illustrates that TPDUs may flow in both directions even when user data flows only from sender to receiver. TPDUs from receiver to sender are examples of *control TPDUs*, which are exchanged between transport entities for connection management. When the flow of user data is bidirectional, control and data information can be *piggybacked*, as discussed in Section 4. Control TPDUs may flow in both directions between sender and receiver, even in the absence of user data.

Figure 2 shows the terminology we use to describe what happens to the request APDU "`GET /research.html`" as it passes through the various layers on its way from the Web client to the Web server. When the user sender submits the request APDU to the transport sender, that APDU becomes a TSDU. The transport sender adds its own header information to the TSDU, to construct a TPDU that it can send to the transport receiver. TPDUs exchanged by the transport entities are *encapsulated* (i.e., contained) in *NPDU*s which are exchanged between the network entities, as illustrated in Figure 2. The network layer routes NPDUs between the local and remote network entities over intermediate links. When an NPDU arrives, the network layer entity processes the NPDU header and passes the payload of the NPDU to a transport layer entity. The transport entity either passes the payload of the TPDU to the transport user if it is user data, or processes the payload itself if it is a control TPDU.

In the previous paragraph we describe a single APDU becoming a single TSDU, being encapsulated in a single TPDU, which in turn becomes a single NSDU encapsulated in a single NPDU. This is the simplest case, and one that is likely to occur for a small APDU such as the HTTP request in our example. However, there are many other possibilities for the relationships between APDUs, TSDUs, TPDUs, NSDUs, and NPDUs, as described in Step (5) of Section 2, and also in Sections 3 and 4.

Figure 2 also shows some of the ter-

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS

Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS

Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS

Sync your system to PACER to automate legal marketing.