The Transport Layer: Tutorial and Survey

SAMI IREN and PAUL D. AMER
Department of Computer and Information Sciences, University of Delaware, Newark, DE 19716
and
PHILLIP T. CONRAD
Department of Computer and Information Sciences, Temple University, Philadelphia, PA 19122

This work 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.

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 systems 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—*applications, protocol architecture (OSI model)*; C.2.5 [**Computer-Communication Networks**]: Local and Wide-Area Networks—*Internet*

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. Essentially, the transport layer isolates applications from the technology, design, and idiosyncrasies 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 some of the best researchers and practitioners in the field. In some sense, 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.

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.

[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.

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).

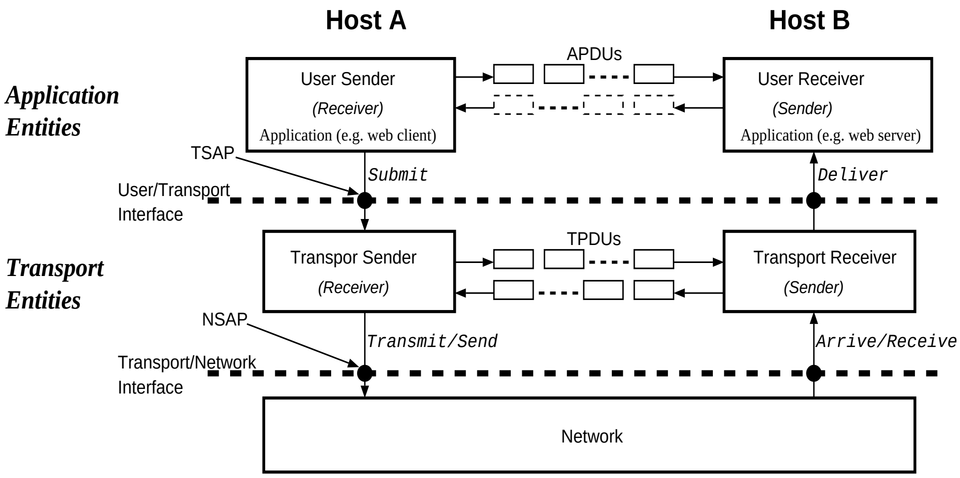

Fig. 1. Transport Service

The term *transport entity* refers to the 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 4.4.3. 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 2.4, and in Sections 3.1, 4.8, 4.9, and 4.10.

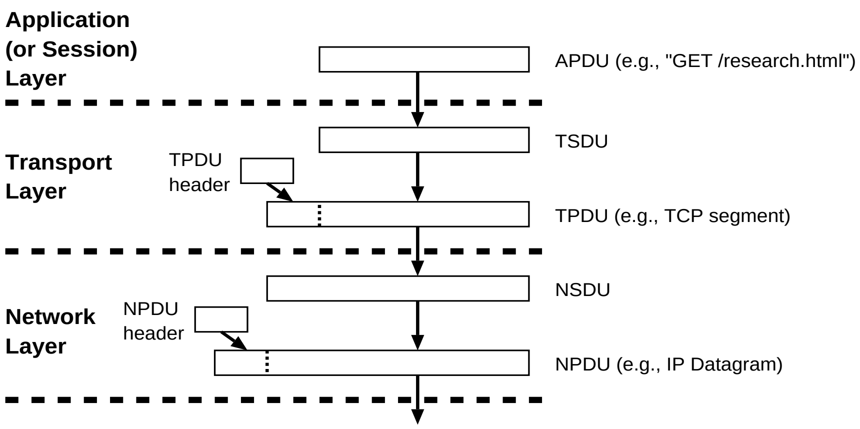

Fig. 2. Relationship between Service Data Units (SDUs) and Protocol Data Units (PDUs).

Figure 2 also shows some of the terminology commonly used in the Internet protocol suite to identify PDUs at various layers. TCP's PDUs are called *segments,* while IP's NPDUs are called *datagrams.* However, the Internet community's terminology can be confusing. For example, the term *packet* is often used informally to refer to both IP Datagrams (NPDUs) and TCP segments (TPDUs). The term *datagram* can refer either to IP's NPDUs, or to the TPDUs of the User Datagram Protocol. To avoid such confusion, we make use of some OSI terms throughout the paper. Although some may criticize OSI terminology as being cumbersome, it often has the advantage of being more precise.

2.4 Example TCP Connection, Step-by-Step

Now let us consider the Web document retrieval example, one step at a time. This discussion omits many details; the purpose here is to provide a general understanding of the transport layer in the context of a well-known application.

(1) You select "`http://www.eecis.udel.edu/research.html`" with your Web client.

—`http` indicates the application layer protocol to be used; more importantly from a transport layer perspective, it also implicitly indicates the *TCP port number* that will be used for the connection (80, in this case). Port numbers differentiate between multiple connection points on the same machine; for example, between servers for Web, File Transfer Protocol (FTP) and telnet (remote terminal) requests.

—"`www.eecis.udel.edu`" indicates the host name to which a TCP connection is established. This is converted to the IP address 128.175.2.17 by the Domain Name System (which is outside the scope of this discussion). The combination of IP address and TCP port number `(128.175.2.17, 80)` represents what OSI calls a *TSAP address*. A TSAP (Transport Service Access Point) address identifies one endpoint of a communication channel between a process on a local machine, and a process on a remote machine.

—"`/research.html`" is the file you are requesting; the Web client uses this to form the HTTP request (APDU) "`GET /research.html`" that must be sent to the Web server via TCP.

(2) The Web client starts by making a *connection request* to the transport entity at `(128.175.2.17, 80)` by calling the `connect()` function. This causes the local-client TCP entity (or simply, the "local TCP") to initiate a *3-way-handshake* with the remote-server TCP entity (or simply, the "remote TCP"). TPDUs are exchanged between the TCP entities to ensure reliable connection establishment, and to establish initial sequence numbers to be used by the transport layer (see Figure 3, and Section 4.3.3). If the 3-way-handshake fails, TCP notifies the application by returning an error code as the result of the `connect()` function; otherwise a success code is returned. In this way, TCP provides what OSI calls *confirmation* of the connect request.

(3) Once the connect request is confirmed, the Web client submits a request to send data (in this case the APDU "`GET /research.html`"). The local TCP then sends this data—most likely, in a single TPDU. This TPDU (i.e., TCP segment) contains the service user's data (the TSDU), plus a transport layer header, containing (among other things) the negotiated initial sequence number for the data. The purpose of this sequence number is discussed in Step 6.

(4) When the remote TCP receives the TPDU, the data "`GET /research.html`" is buffered. It is delivered when the Web server does a `read()`. In OSI terminology, this delivery is known as a *data indication*. The remote TCP also sends back an *acknowledgment* (ACK) control TPDU to the local TCP. Acknowledgments inform the transport sender about TPDU arrivals (see Section 4.4).

(5) The Web server then responds with the contents of "`research.html`". This file may be too large to be efficiently submitted to TCP in one `write()` call, i.e., one TSDU. If so, the Web server divides this APDU into multiple `write()` calls, i.e., multiple TSDUs. The remote TCP then sends these TSDUs to the local TCP in multiple TPDUs, utilizing the service of the underlying network layer.

One interesting aspect of TCP is that the sizes of the TSDUs submitted by the transport entities may bear little or no relationship to the sizes of the TPDUs actually exchanged by TCP. TCP treats the flow of data from sender to receiver as a *byte-stream* and segments it in whatever manner seems to make best use of the underlying network service, without regard to TSDU boundaries. It delivers data in the same fashion; thus for TCP, the boundaries between APDUs, submitted TSDUs, TPDUs, and delivered TSDUs may all be different. Other transport services/protocols have a *message* orientation, meaning that they preserve the boundaries of TSDUs (see Section 3.1.)

(6) Because the network layer (IP in this case) can lose or reorder NSDUs, TCP must detect and recover from network errors. As the remote TCP sends the TPDUs containing the successive portions of the file "`research.html`", it includes a sequence number in each TPDU corresponding to the sequence number of the first byte in that TPDU relative to the entire flow of data from remote TCP to local TCP. The remote TCP also places a copy of the data in each TPDU sent into a buffer, and sets a timer. If this timer expires before the remote TCP receives a matching ACK TPDU from the local TCP, the data in the TPDU will be retransmitted in a new TPDU. In TCP, the data are tracked by individual byte-stream sequence numbers, and TPDUs retransmitted may or may not correspond exactly to the original TPDUs. The remote TCP also places a *checksum* in the TPDU header to detect bit errors introduced by noise or software errors in the network, data link and physical layers. Such *error control* is a major function of the transport layer as discussed in Section 4.4.

(7) As TPDUs are received by the local TCP, any TPDUs with checksum errors are discarded. The sequence numbers of the incoming TPDUs are checked to ensure that no pieces of the byte-stream are missing or have arrived out-of-order. Any out-of-order pieces will be buffered and re-ordered. As TPDUs arrive, the local TCP responds to the remote TCP with ACK TPDUs. If ACKs are lost, the remote TCP may send a TPDU which duplicates data sent in an earlier TPDU. The local TCP also discards any duplicate data that arrives. By detecting and recovering from loss, reordering, duplication and noise, the local and remote TCP entities cooperate to ensure reliable delivery of data. As pieces of the byte-stream arrive correctly, these will be buffered until the Web client requests them by doing `read()` calls. Each `read()` results in delivery of a TSDU to the Web client.

(8) In general, a TCP connection is bidirectional, and either side may initiate the closing of the connection. However, in first-generation Web systems, the server initiates the close by calling the `close()` function after it sends the last piece of the file requested. In OSI terminology, this is called a *disconnect request.*

TCP handles disconnect requests with a *4-way-handshake* procedure (shown in Figure 4) to ensure graceful termination without loss of data. The remote TCP responds to the server's disconnect request by sending a disconnect TPDU and waiting for an ACK TPDU. The local TCP signals the server's disconnect request to the client by returning "end-of-file" as the result of a client `read()` request. The client then responds with its own `close()` request; this causes another exchange of a disconnect TPDU and ACK. This and other methods of connection termination are discussed in Section 4.3.4.

2.5 What this example shows... and does not show

From this example, we see that TCP provides a connection-oriented (CO) byte-stream, no-loss, no-duplicates, ordered transport service, and that its protocol is designed to operate on top of a lossy network service, such as IP datagram service [Postel 1981].

This example also shows a few aspects of communication between the user layer and the transport layer, and between peer entities. It illustrates one example of connection establishment, reliable data exchange, and connection termination. However, many aspects of this example were not discussed, such as flow control and congestion avoidance, piggybacking of ACKs, and round-trip-time estimation, to name just a few.

Also, consider that in the case of a Web browser using TCP, the transport service is *reliable*, which means (among other things) that the transport service may not lose any TSDUs. This guarantee comes at the expense of potentially increased delay, since reliable delivery may require more TPDUs to be exchanged. Other transport services (for example, those used for audio or video transmission) do not provide reliability, since for some applications lower delay is more important than receiving every TSDU.

Many variations exist both in the type of service provided by the transport layer, and in the design of transport protocols. These variations are what make the transport layer so interesting! Sections 3 and 4 provide an overview of the most important of these variations.

3. TRANSPORT SERVICE

This section classifies the typical *service* provided by the transport layer (see Table 1). A transport service abstracts a set of functions that is provided to a higher layer. A *protocol*, on the other hand, refers to the details of how a transport sender and a transport receiver cooperate to provide that service. In defining a service, we treat the transport layer as a black box. In Section 4, we look at internal details of that black box.

Table 1. Transport Service Features

CO-message vs. CO-byte vs. CL
No-loss vs. Uncontrolled-loss vs. Controlled-loss
No-duplicates vs. Maybe-duplicates
Ordered vs. Unordered vs. Partially-ordered
Data-integrity vs. No-data-integrity vs. Partial-data-integrity
Blocking vs. Non-blocking
Multicast vs. Unicast
Priority vs. No-priority
Security vs. No-security
Status-reporting vs. No-status-reporting
Quality-of-service vs. No-quality-of-service

One of the valuable contributions of the OSI Reference Model is its emphasis in distinguishing between service and protocol. This distinction is not well differentiated in the TCP/IP protocol suite where RFCs intermix service and protocol concepts, frequently in a confusing manner. As will be seen in Section 4, there are often several different protocol mechanisms that can provide a given service.

3.1 CO-message vs. CO-byte vs. CL

Transport services can be divided into two types: *connection-oriented* and *connectionless*. A connection-oriented (CO) service provides for the establishment, maintenance, and termination of a logical connection between transport users. The transport user generally performs three distinct phases of operation: connection establishment, data transfer, and connection termination. When a user sender wants to transmit some data to a remote user receiver by using a connection-oriented service, the user sender first explicitly asks the transport sender to open a connection to the remote transport receiver (*T-Connect*). Once a connection is established, the user sender provides the transport sender with the data to be transmitted (*T-Data*). At the end of the data transfer, the user sender explicitly asks the transport sender to terminate the connection (*T-Disconnect*).

CO service itself has two variations: *message-oriented* and *byte-stream*. In the former, the user sender's messages (or what OSI calls *service data units* (SDUs)) have a specified maximum size, and message boundaries are preserved. When two $1K$ messages are submitted by a user sender, they are delivered to the user receiver as the same two distinct $1K$ messages, never for example as one $2K$ message or four $.5K$ messages. TP4 is an example that provides message-oriented service. In byte-stream service as provided by TCP, the flow of data from user sender to user receiver is viewed as an unstructured sequence of bytes that flow in a FIFO manner. There is no concept of message (or message boundary or maximum message size) for the user sender. Data submitted by a user sender is appended to the end of the byte-stream, and the user receiver reads from the head of the byte-stream.

An advantage of byte-stream service is that it gives the transport sender flexibility to divide TSDUs in ways that make better use of the underlying network service. However byte-stream service has been criticized because it does not deliver data to an application in meaningful units. A message-oriented service better preserves the semantics of each APDU down through the lower layers for so-called Application Level Framing [Clark and Tennenhouse 1990] (see Section 6). One research team notes that adaptive applications are much easier to develop if the unit of processing (i.e., APDU) is strongly tied to the unit of control (i.e., TPDU) [Dabbous and Diot 1995].

A connectionless (CL) service provides only one phase of operation: data transfer. There are no *T-Connect* and *T-Disconnect* primitives exchanged (either explicitly or implicitly) between a user sender and the transport sender. When the user sender has some data, it simply submits the data to the transport sender. As with CO-message service, in a CL service the user sender submits messages, and message boundaries are maintained. Messages submitted to a CL service as in UDP often are referred to as *datagrams*.

3.2 Reliability

Unfortunately the terms *reliable* and *unreliable* mean different things to different people. To avoid ambiguity in our definition, unless otherwise stated, a service is reliable if and only if it is all of the following (as defined below): *no-loss*, *no-duplicates*, *ordered*, and *data-integrity*. If even one feature is lacking, the service is unreliable.

3.2.1 *No-loss vs. Uncontrolled-loss vs. Controlled-loss.* There are three levels of delivery guarantee. A *no-loss* (or *at-least-once delivery*) service guarantees that for all data submitted to the transport sender by the user sender, one of two results will occur. Either (1) the data is delivered[2] to the user receiver, or (2) the user sender is notified that some data may not have been delivered. It never occurs that a user sender believes some data was delivered to the user receiver when in fact it was not. For example, TCP provides a no-loss service using disconnection as a means of notifying a user sender that data may not have been delivered.[3]

An *uncontrolled-loss* (or *best-effort*) service does not provide the above assurance for any of the data. For example, UDP provides an uncontrolled-loss service. Any data submitted to UDP by a user sender may fail to be delivered to the user receiver without the user sender being notified. Analogously, the default service of the United States Postal Service is an uncontrolled-loss service. A user sender may not be notified if a letter is not delivered to the user receiver.

A *controlled-loss* service lies in between no-loss and uncontrolled-loss. Loss may occur, but there is control over the degree of loss. Several variations of a controlled-loss service are possible. For example, a user sender might request different loss levels for different messages. The k-XP protocol [Amer et al. 1997; Marasli et al. 1996] supports three controlled-loss service classes:

- reliable messages will be retransmitted (if needed) until successfully delivered to the user receiver;
- partially reliable messages will be retransmitted (if needed) at most k times and then dropped if unsuccessful, where k is a user sender parameter; and
- unreliable messages will be transmitted only once.

In the TRUMP protocol [Golden 1997], a user sender assigns a time limit to each message. The transport layer does its best to deliver the message before the time limit; if unsuccessful, the message is discarded.

3.2.2 *No-duplicates vs. Maybe-duplicates.* A *no-duplicates* (or *at-most-once delivery*) service (e.g., TCP) guarantees that all data submitted to the transport sender will be delivered to the user receiver at most once.

A *maybe-duplicates* service (e.g., UDP) does not provide the above guarantee. Efforts by the protocol may or may not be made to avoid delivering duplicates, but delivery of duplicates may occur nevertheless.

---

[2]Depending on the data-integrity service defined in Section 3.2.4, delivered data may or may not contain bit errors.

[3]Strictly speaking, for the most common TCP API, namely the Berkeley Sockets API, this is only true if the application specifies the SO_LINGER socket option; by default, it is possible for a TCP connection to lose data without notifying the application during connection teardown [Stevens 1998, p.187].

3.2.3 *Ordered vs. Unordered vs. Partially-ordered.* An *ordered* service (e.g., TCP) preserves the user sender's submission order of data when delivering it to the user receiver. It never occurs that a user sender submits two pieces of data, first A, then B, and A is delivered *after* B is delivered.

An *unordered* service (e.g., UDP) does not provide the guarantee above. While the service may try to preserve the order of data when delivering it to the user receiver, no guarantee of preservation-of-order is made.

A *partially-ordered* service guarantees to deliver pieces of data in one of a set of permitted orders as predefined by a partial order relation agreed upon by the user sender and user receiver. Partially-ordered service is useful for applications such as multimedia communication or distributed databases where delivery of some objects is constrained by others having been delivered, yet independent of certain other objects arriving [Connolly et al. 1994]. POC is an example protocol providing partially-ordered service [Conrad et al. 1996].

3.2.4 *Data-integrity vs. No-data-integrity vs. Partial-data-integrity.* A *data-integrity* service ensures with *high probability* that all data bits delivered to a user receiver are identical to those originally submitted. The actual probability achieved in practice depends on the strength of the error detection method. Good discussions of error detection methods can be found in [Lin and Costello 1982; McNamara 1998]. TCP uses a 16-bit checksum for data-integrity; its goodness is evaluated in [Partridge et al. 1995].

A *no-data-integrity* service does not provide any guarantees regarding bit errors. Any number of bit errors may occur in the delivered data. UDP may or may not be configured to use a checksum; in the latter case, UDP provides a no-data-integrity service.

A *partial-data-integrity* service allows a controlled amount of bit errors in delivered data. This service may be acceptable as a means of achieving higher throughput. A real-time multimedia application might request the transport receiver to tolerate certain levels of bit errors when errored data could be useful to the user receiver [Han and Messerschmitt 1996]. For example, a Web browser retrieving an image might prefer to progressively display a partially damaged image while waiting for the retransmission and delivery of the correct data (assuming the decompression algorithm can process damaged image data.)

3.2.5 *Remarks on Reliability and CO vs. CL.* Note that all four aspects of reliability – loss, duplicates, order, and data-integrity – are *orthogonal*; they are independent functions. For example, *ordered service* does not imply *no-loss service*; some portion of the data might get lost while the data that is delivered is guaranteed to be in order.

A wide range of understanding (i.e., confusion) exists in the relationship between a service being CO or CL and whether or not it is reliable. These two services are orthogonal, yet so many people presume a CO service is reliable. Why? In the past when the number of available services and protocols was smaller, and applications were fewer and less diverse in their service needs, reasoning about connections and reliability was simple:

Whereas:	TCP service is CO and TCP service is reliable,
Whereas:	TP4 service is CO and TP4 service is reliable,
Whereas:	X.25 service is CO and X.25 service is reliable,
Therefore:	CO service $\equiv$ reliable service.

Whereas:	UDP service is CL and UDP service is unreliable,
Therefore:	CL service $\equiv$ unreliable service.

We emphasize that **these equivalences do not hold!** It is quite reasonable to design a transport layer providing CO service that is not reliable (e.g., provides controlled-loss rather than no-loss service, or unordered rather than ordered service), say to support best-effort compressed-image transmission [Iren et al. 1998; Iren]. Similarly a transport layer can provide a CL service that is reliable if an underlying network layer is reliable.

3.3 Blocking vs. Non-Blocking

A *blocking* service ensures that the transport layer is not overwhelmed with incoming data. In this case, a user sender submits data to the transport sender and then waits for the transport sender to signal that the user sender can resume processing. A blocking service provides flow control between user sender and transport sender.

A *non-blocking* service allows the user sender to submit data and continue processing without awaiting the transport sender's ok. A non-blocking service does not take into account the transport layer's buffering capabilities, or the rate at which the user receiver consumes data. The user sender submits data at any rate it chooses, possibly resulting in either data loss or notification to the user sender that data cannot be submitted right now.

Of the many transport protocols surveyed, all could potentially provide blocking or non-blocking service depending on their API; one can therefore argue that this service feature is an implementation decision, not an inherent feature of a given transport layer. (Note that the term "blocking" as used here has no relationship to its use in Section 4.9.)

3.4 Multicast vs. Unicast

A *multicast* service enables a user sender to submit data, a copy of which will be delivered to *one or more* user receiver(s). Applications such as real-time news and information broadcasting, teleconferencing, and remote collaboration can take advantage of a multicast service to reach multiple destinations. Depending on the reliability of the service, delivery to each user receiver may be no-loss vs. uncontrolled-loss vs. controlled-loss, no-duplicates vs. maybe-duplicates, etc. The use of a multicast service influences many of the protocol features to be discussed in Section 4.

A *unicast* service limits the data submitted by a user sender to be delivered to *exactly one* user receiver.

(As stated earlier, although the tables in the appendix indicate which protocols surveyed offer multicast service, this paper does not survey or discuss multicast protocols.)

3.5 Priority vs. No-priority

A *priority* service enables a user sender to indicate the relative importance of various messages. A user of priority service may expect higher priority data to be delivered sooner, if possible, than lower priority data, even at the expense of slowing down data submitted earlier. When combined with uncontrolled-loss or controlled-loss service, a priority service may drop lower priority data when necessary, thereby allowing the delivery of higher priority data with smaller delay and/or higher probability. Priority also may be passed down to a network service if it has a notion of priority, such as IP's type of service field.

In OSI, priority service is provided in the form of *expedited data* and the priority QoS parameter (see Section 3.8). Expedited data is an entirely separate data flow that is not subject to normal data flow control.

A *no-priority* service does not allow a user sender to differentiate the importance of classes of data.

Despite common misconceptions to the contrary, TCP's *urgent data* service is neither a priority service nor an expedited data service. The Berkeley Sockets API provides access to this service via what it calls *out-of-band* data. However, this is misleading, since data sent in urgent mode is *not* expedited, sent out-of-band, or sent with higher priority. Rather, TCP urgent mode is a service by which the user sender (i.e., an application) marks some portion of the byte-stream as needing special treatment by the user receiver. Thus, *signaling* the presence of urgent data and *marking* its position in the data stream are the only aspects that distinguish the delivery of urgent data from the delivery of all other TCP data; for all other purposes, urgent data is treated identically to the rest of the TCP byte-stream. The user receiver must read every byte of data exactly in the order it was submitted regardless of whether or not urgent mode is used.

3.6 Security vs. No-security

A *security* service (e.g., TP4) provides one or more security functions such as: authentication, access control, confidentiality, and integrity [ISO 1989]. *Authentication* is the verification of user sender's and user receiver's identities. *Access control* checks a user's permission status before allowing the use of different resources. *Confidentiality* guarantees that only the intended user receiver(s) can decode and understand the user sender's data; no other user receiver can understand the data's content. Finally, *integrity*[4] detects (and sometimes recovers from) any modification, insertion, deletion, or replay of transport sender's data. *Secure routing* also has been noted as a security service [Stallings 1997, p.591]. This service guarantees that while going from user sender to user receiver, data will only pass through secure links and secure intermediate routers.

A *no-security* service does not provide any of the above security functions. TCP currently provides no-security-service, although some discussion is underway to include security features in a future generation.

[4]See Section 3.2.4 for a slightly different usage of integrity.

3.7 Status-reporting vs. No-status-reporting

A *status-reporting* service allows a user sender to obtain specific information about the transport entity or its connections. This information may allow the user sender to make better use of the available service. Some information that a status reporting service might provide are: performance characteristics of a connection (e.g., throughput, mean delay), addresses (network, transport), current timer values, or the state of the protocol machine supporting a connection.

A *no-status-reporting* service does not provide any information about the transport entity and its connections.

A distinction can be made between status-reporting and QoS monitoring as, for example, in applications based on the Real-time Transport Protocol (RTP). Although QoS monitoring provides some of the information listed in the above definition (delay, throughput), it provides no information on the protocol's internal structure such as state information, timer values, etc. Status-reporting service provides explicit interactions between a user and its transport entity. In QoS monitoring, there are no such explicit interactions.

Of the protocols surveyed, only TCP provides something resembling status-reporting service. When TCP's *Socket Debug Option* is enabled, the kernel maintains for future analysis a trace record of what happens on a connection [Stevens 1994, p.496]. Status-reporting is a valuable service that designers should consider incorporating into their protocol implementations.

3.8 Quality-of-service vs. No-quality-of-service

A transport layer that provides *Quality of Service* (QoS) allows a user sender to specify the quality of transmission service desired. The protocol then presumably optimizes network resources to provide the requested QoS. Since the underlying network places limits on the service that can be provided by the transport protocol, a user sender should recognize two facts: (1) depending on the underlying network's capabilities, a transport protocol will have varying degrees of success in providing the requested QoS, and (2) there is a trade-off among QoS parameters such as reliability, delay, throughput, and cost of service [Marasli et al. 1996].

Transport QoS is a broad and complicated issue. No universally accepted list of parameters exists, and how the transport layer should behave for a desired QoS under different circumstances is unclear. The ISO Transport Service [ITU-T 1995c] defines a number of possible performance QoS parameters that are negotiated during connection establishment. User senders can specify (sustained) target, acceptable, and minimum values for various service parameters. The transport protocol examines these parameters, and determines whether it can provide the required service; this depends in part on the available network service. ISO specifies eleven QoS parameters:

- *Connection Establishment Delay* is the maximum acceptable time between a transport connection being requested and its confirmation being received by the user sender.
- *Connection Establishment Failure Probability* is the probability a connection cannot be established within the maximum connection establishment delay time due to network or internal problems.

- *Throughput* is the number of bytes of user sender data transferred per unit time over some time interval.
- *Transit Delay* is the elapsed time between a message being submitted by a user sender and being delivered to the user receiver.
- *Residual Error Rate* is the ratio of incorrect, lost, and duplicate TSDUs to the total number of TSDUs that were sent.
- *Transfer Failure Probability* is the ratio of total transfer failures to total transfer samples observed during a performance measurement.
- *Connection Release Delay* is the maximum acceptable time between a transport user initiating release of a connection and the actual release at the peer transport service user.
- *Connection Release Failure Probability* is the fraction of connection release attempts that did not complete within the agreed upon connection release delay interval.
- *Protection* is used by the user sender to specify interest in having the transport protocol provide protection against unauthorized third parties reading or modifying the transmitted data.
- *Priority* allows a user sender to specify the relative importance of transport connections. In case of congestion or the need to recover resources, lower-priority connections are degraded or terminated before the higher-priority ones.
- *Resilience* is the probability that the transport protocol itself will spontaneously terminate a connection due to internal or network problems.

A transport layer that provides *No-Quality-of-Service* (No-QoS) does not allow a user sender to specify desired quality of transmission service.

The original definitions of TP0 and TP4 support QoS service, however, only recently have the lower layers of networks matured sufficiently for the community to consider transport layer QoS handling. The ATM environment supports only two QoS parameters: (sustained) target, acceptable, and minimum throughput and transit delay [ITU-T 1995a]. Within the Internet IETF community, much work is ongoing to create an integrated services (INTSERV) QoS control framework so that the TCP/IP suite can provide more than its current no-QoS service. While not itself a transport protocol, RSVP [Braden et al. 1997] sits on top of IP. RSVP allows a host to request specific qualities of service from the network. A request may refer to a single path in the case of unicast or multiple paths in the case of multicast. General QoS parameters supported by the INTSERV framework are defined in [Shenker and Wroclawski 1997]. Requests result in reserved resources by individual network elements (i.e., subnets, IP routers) along the path. Thus far use of RSVP has been defined to provide two kinds of Internet QoS: controlled load [Wroclawski 1997] and guaranteed [Shenker et al. 1997].

4. TRANSPORT PROTOCOL FEATURES

The previous section describes general features of transport layer service. This section focuses on internal transport layer mechanisms that provide this service. It is important to understand that almost every service can be accomplished by several different protocols.

4.1 Connection-oriented vs. Connectionless

Not only can a service be classified CO or CL, so too can a protocol that provides either service. The distinction depends on the establishment and maintenance of *state information*, a record of characteristics and events related to the communication between the transport sender and receiver. Perhaps the most important piece of state information is the sequence number that identifies a TPDU. Consistent viewpoints of the sequence numbers used by transport sender and transport receiver are required for reliable data transfer. Choosing an initial value for the sequence number can be hazardous, particularly when two transport entities close and immediately reopen a connection (see Section 4.4.7).

A transport protocol is *CO* (e.g., TCP) if state information is maintained between transport entities. Typically, a CO protocol has three phases: connection establishment, data transfer, and connection termination. In the connection establishment phase, state information is initialized either explicitly by exchanging control TPDUs, or implicitly with the arrival of first data TPDU. During a connection's lifetime (i.e., data transfer phase), state information is updated to reflect the reception of data and control PDUs (e.g., ACKs). When both transport entities agree to terminate the connection, they discard the state information. Note the distinction between a CO service and a CO protocol. A CO service entails a three phase operation to establish a logical connection between transport *users*. A CO protocol entails a three phase operation to maintain state information between transport *entities*.

If no state information is maintained at the transport sender and receiver, the protocol is *CL*. A CL protocol is based on individually self-contained units of communication often called *datagrams* that are exchanged independently without reference to each other or to any shared state information. Each datagram contains all of the information that the receiving transport entity needs to interpret it.

4.2 Transaction-oriented

Transaction-oriented protocols attempt to optimize the case where a user sender wishes to communicate a single APDU (called a request) to a user receiver, who then normally responds with a single APDU (called a response). Such a request/response pair is called a transaction.

A transaction-oriented transport protocol attempts to provide *reliable* service for transactions with as few TPDUs as possible, ideally only one for the request and one for the response. This is done by trying to minimize overhead for connection establishment. Transactions share the following characteristics [Braden 1992a]: an asymmetrical model (i.e., client and server), simplex data transfer, short duration, low delay, few data TPDUs, message orientation, and the need for no-duplicates service. Examples of transaction-oriented protocols include VMTP and T/TCP, a backwards compatible TCP extension.

4.3 CO Protocol Features

The following subsections refer only to CO protocols.

4.3.1 *Signaling.* *Signaling* is the exchange of control (i.e., state) information between transport entities for managing a connection.[5] Signaling is used to establish and terminate a connection and to exchange communication system parameters. Signaling can be accomplished *in-band*, where control information and user data are multiplexed on the same connection, or *out-of-band*, where separate connections are used. In-band signaling (e.g., TCP) is generally more suitable for applications that require short-lived connections (e.g., transaction-oriented communications). The extra overhead incurred for managing two bands is avoided and small amounts of user data can be transmitted during the signaling operation.

Out-of-band signaling (e.g., TP++) is desirable for high-speed communication systems. It avoids the overhead of separating signaling information from user data by doing so below the transport layer. Out-of-band signaling can support transporting more than one kind of data over a connection thereby facilitating operations involving third parties, for example, a server validating security information or billing.

4.3.2 *Unidirectional vs. Bidirectional.* In a *unidirectional* connection (e.g., NETBLT, VMTP) data flows only in one direction at a time (half duplex). That is, while one transport entity is sending data, the other may not send data in the reverse direction. Each transport entity can assume the role of transport sender or transport receiver, but not both simultaneously.

In a *bidirectional* connection (e.g., TCP), both transport entities simultaneously assume the roles of transport sender and transport receiver thereby allowing full duplex data flow.

4.3.3 *Connection Establishment.* Three modes of connection establishment are shown in Figure 3. The cost associated with connection establishment can be amortized over a connection's lifetime. For short-lived connections which result from transaction-oriented applications, this cost can be significant. Therefore, protocols for short-lived connections have been designed using a timer-based connection establishment and termination mechanism. For longer connections and when avoiding false connections is important, 2-way or 3-way *handshake* mechanisms are needed. A handshake involves the explicit exchange of control TPDUs.

In *implicit* connect, the connection is open as soon as the first TPDU is sent or received. The transport sender starts transmitting data without any explicit connection verification by the transport receiver. Further data TPDUs can be sent without receiving an ACK from the transport receiver. Implicit connections can provide reliable service only if the protocol definition guarantees that delayed TPDUs from any previously closed connection cannot cause a false open. T/TCP [Braden 1994] achieves this via connection-unique identifiers and timer-based connection termination.

In *2-way-handshake* connect (e.g., APPN, SSCOP/AAL5), a *CR-TPDU* (Connection Request) and a *CC-TPDU* (Connection Confirm) are exchanged to establish the connection. The transport sender may transmit data in the CR-TPDU, but sending additional data is prohibited until the CC-TPDU is received. During this

[5]While in some cases data may be exchanged, signaling generally refers to exchanging control info.

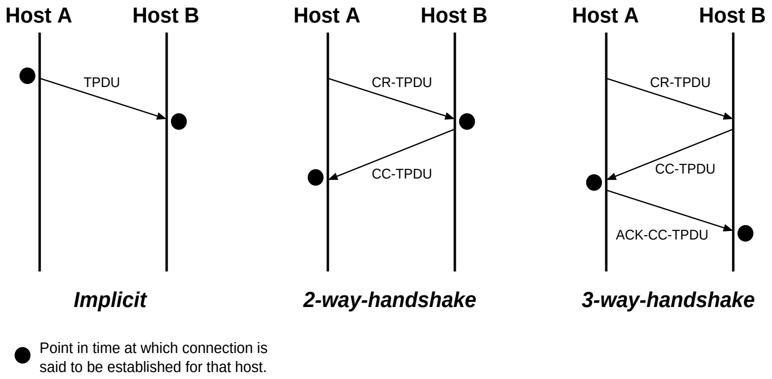

Fig. 3. Three modes of connection establishment

handshake, QoS parameters such as buffer size, burst size, burst rate, etc., can be negotiated. When the underlying network service provides a small degree of loss, a 2-way-handshake mechanism may be good enough to establish new connections without significant risk of false connections.

In *3-way-handshake* connect (e.g., TCP), the transport sender sends a *CR-TPDU* to the transport receiver which responds with a *CC-TPDU*. The procedure is completed with an *ACK-CC-TPDU* (ACK for Connection Confirm). Normally no user data is carried on these connection establishment TPDUs.[6] If the underlying network service provides an unacceptable degree of loss, a 3-way-handshake is needed to prevent false connections that might result from delayed TPDUs. Because TCP was designed specifically for use over an unreliable network service, TCP uses a 3-way handshake.

4.3.4 *Connection Termination.* Four modes of connection termination are shown in Figure 4.

With *implicit* disconnect (e.g., VMTP), when a transport entity does not hear from its peer for a certain time period, the entity terminates the connection. Typically, implicit disconnection is used with implicit connection establishment.

In *abortive* disconnect, when a transport entity must close the connection abnormally due to an error condition, it simply sends an *abort-TPDU* and terminates the connection. The entity does not wait for a response. Thus TPDUs in transit in either direction may be lost.

In *2-way-handshake* disconnect (e.g., NETBLT), a transport entity sends a *DR-TPDU* (Disconnect Request) to its peer and receives a *DC-TPDU* (Disconnect Confirm) in return. If a connection is unidirectional (see Section 4.3.2), the transport sender usually initiates connection termination and before discarding its state information verifies the reception of all TPDUs by the transport receiver. In a bidirectional connection, a 2-way-handshake can only verify reception of TPDUs in

[6]TCP allows user data to be sent on a *CR-TPDU*. However, that data cannot be delivered to the user until the 3-way-handshake is completed. Furthermore, outside of T/TCP compliant implementations, TCP implementations typically do not take advantage of this feature [Stevens 1996].

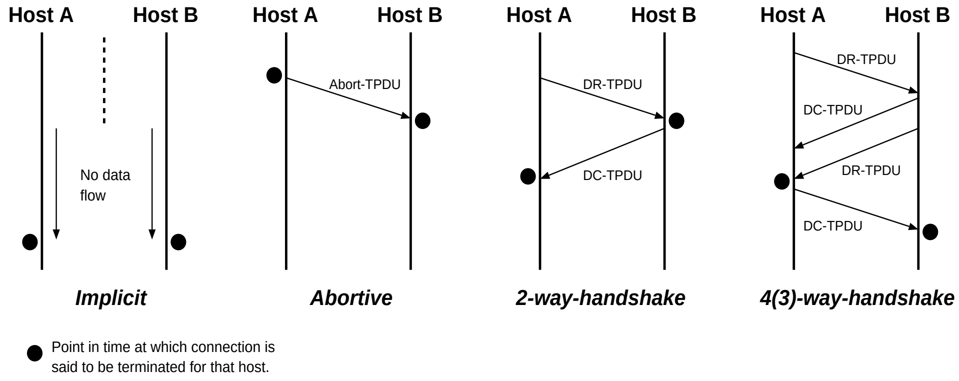

Fig. 4. Four modes of connection termination

one direction. TPDUs in transit in the reverse direction may be lost.

Finally, in *4(3)-way-handshake* disconnect (e.g., TCP), two 2-way-handshakes are used, one for each direction of data flow. The transport entity that closes its sending flow sends a *DR-TPDU* to its peer entity (in TCP the FIN flag is set in its last TPDU). This disconnect request is then acknowledged by the transport receiver as soon as all preceding TPDUs have been received. The connection is terminated when data flows in both directions are closed. The number of control TPDUs exchanged can be reduced to three if the first *DC-TPDU* also functions as a *DR-TPDU* for the reverse direction.

Connections may be terminated either *gracefully* or *ungracefully*. In an ungraceful termination as in TP0 or TP4[7], some data in transit may be lost. In a graceful close, no data in transit is lost since a connection is closed only after all data have arrived at their destination. For bidirectional connections, only a 4(3)-way-handshake can guarantee graceful close.

4.4 Error Control: Sequence Numbers, Acks, and Retransmissions

Error control is a combination of techniques to guard against loss or damage of user data and control information. CL network protocols such as IP perform no error control on user data. Even CO networks are being designed with lightweight virtual circuits that provide no error control [McAuley 1990; Parulkar and Turner 1989]. Recent studies have shown that for realistic high-speed networks with low error rates, transport layer error control is more efficient than link layer error control [Bae et al. 1991; Bhargava et al. 1988]. There are two phases in transport layer error control: *error detection* and *error reporting and recovery*.

4.4.1 *Error Detection.* Error detection identifies lost, misordered, duplicated, and corrupted TPDUs. *Sequence numbers* help uncover the first three problems. Corrupted data is discovered by means of (1) *length fields* that allow a transport receiver to detect missing or extra data, and (2) redundant information in the form of *error detecting codes (EDC)*. Cyclic redundancy checks and other forms

[7]The OSI Session Layer provides the graceful close.

of checksums are the most common examples of EDCs. An EDC can verify the header/trailer, the data, or both. TCP and AAL5 contain a single checksum on both. XTP performs one checksum on the header and allows for an optional second checksum on the data.

Separate EDCs for header and data are recommended for multimedia applications [La Porta and Schwartz 1991]. In voice applications, where corrupted data is tolerable but retransmissions are less advantageous due to delay constraints, only a header EDC needs to be used. In image transfer using both EDCs, corrupted data may be used temporarily until a corrected version is obtained via retransmission.

4.4.2 *Error Reporting and Recovery.* *Error reporting* is a mechanism where the transport receiver explicitly informs the transport sender about errors that have been detected. *Error recovery* is a mechanism used by both transport sender and receiver to recover from errors whether or not they are explicitly reported.

Error reporting and recovery are traditionally accomplished using timers, sequence numbers and acknowledgments. Sequence numbers are assigned to TPDUs (or associated with the byte-stream). The transport receiver informs the transport sender via ACKs about TPDU arrivals so that the sender can retransmit those that are missing.

A *positive* ACK or *PACK* contains sequence number information about those TPDUs that have been received. A *negative* ACK or *NACK*, often known as a *selective reject*, explicitly identifies TPDUs that have not been received.[8] Protocols that use ACKs are known as *Positive Ack with Retransmission* (PAR) or *Automatic Repeat reQuest* (ARQ) schemes. Upon receipt of an ACK, the transport sender updates its state information, discards buffered TPDUs that are acknowledged, and retransmits any TPDUs that are not acknowledged. A protocol that only use PACKs has no error reporting mechanism.

If a transport sender does not receive an ACK within a reasonable timeout period, it may assume something has gone wrong and retransmits unacknowledged TPDU(s). With delays within most underlying networks being variable and dynamic, accurate calculation of a reasonable timeout value within the transport layer is a most difficult problem. However, accurate round-trip time (RTT) estimation is essential [Zhang 1986]. The basic idea is to keep an RTT estimate by measuring the time between sending a TPDU and receiving its ACK. Karn and Partridge showed the importance of not updating this estimate for retransmitted TPDUs, since it is impossible to determine whether the received ACK is for the original TPDU transmission or a subsequent retransmission [Karn and Partridge 1987]. Jacobson refined TCP's retransmission timeout calculation, by using a low-pass filter to estimate the mean, and mean deviation of the RTT [Jacobson 1988]. RTT estimation has been a key area of research in TCP; a informative summary appears in [Stevens 1994, p.299ff].

4.4.3 *Piggybacking.* When a TPDU arrives at a transport receiver, instead of immediately returning an ACK as a separate control TPDU, some protocols (including TCP) artificially delay returning an ACK hoping the user receiver will soon

[8]For clarity in this paper, PACK refers only to a positive ACK, and ACK refers to either a PACK or NACK.

submit its next message to be sent as part of the reverse direction data flow. When this occurs, the ACK is *piggyback*-ed as header information on the reverse direction data TPDU. This is an example of transport layer concatenation (see Section 4.8).

4.4.4 *Cumulative vs. Selective Acknowledgment.* A common type of PACK is the *cumulative* PACK. Each cumulative PACK carries a sequence number indicating that all TPDUs with lower[9] sequence numbers have been received. With cumulative PACKs, even if $PACK_i$ is lost, the arrival of a later $PACK_j$ such that $j > i$, positively acknowledges all TPDUs up to $TPDU_j$. The later cumulative PACK incorporates the information of the previously lost one.[10]

A disadvantage of cumulative PACKs occurs when $TPDU_i$ is lost and $TPDU_{i+1}$, $TPDU_{i+2}$, ..., arrive intact and are buffered. Each PACK following the loss can acknowledge only TPDUs up to and including $TPDU_{i-1}$. Thus, the transport sender may not receive timely feedback on the success of TPDUs sent after $TPDU_i$, and retransmit them unnecessarily.

The second PACK type in its most basic form is the *selective* PACK which acknowledges exactly one TPDU. With selective PACKs, a transport receiver informs the transport sender about each TPDU that arrives successfully, so the transport sender can more likely retransmit only those TPDUs that have actually been lost. Selective PACKs are used by NETBLT and VMTP.

A *block* PACK is a variation of selective PACK where blocks of individual TPDUs are selectively acknowledged [Brown et al. 1989]. For example, an XTP ACK is actually a series of block PACKs. If TPDUs {*1, 2, 4, 5, 6, 9, 10*} arrive and {*3, 7, 8*} are missing, then the block PACK would look like *1-2;4-6;9-10*.

In part due to the influence of innovative acknowledgment schemes in protocols such as XTP, NETBLT, and VMTP, *TCP with selective acknowledgment* (called SACK) recently has been proposed. SACK combines selective (block) and cumulative PACKs within TCP [Braden and Jacobson 1988; Floyd 1996; Mathis et al. 1996]. One simulation study shows the strength of TCP implementations with vs. without selective PACKs [Fall and Floyd 1996].

4.4.5 *Retransmission Strategies.* When a transport sender transmits $TPDU_i$ and determines later that it may not have arrived at the transport receiver, two *retransmission strategies* are possible. (This determination can result when the transport sender does not receive a PACK within a predetermined timeout period, or when it receives back-to-back cumulative PACKs that are identical.) A conservative approach has the transport sender retransmit *selectively* only $TPDU_i$ and wait for a PACK with sequence number larger than previous PACKs. This *selective repeat* approach is used in SNR [Doshi et al. 1993] and avoids retransmitting correctly received TPDUs [Feldmeier and Biersack 1990].

A more aggressive transport sender retransmits $TPDU_i$ and all TPDUs already sent after $TPDU_i$. This *Go-Back-N* approach is fairly simple but decreases channel utilization by potentially retransmitting correctly-received TPDUs. These unnecessary retransmissions then add to network congestion. The protocol SMART

[9] This is TCP's definition; some protocols use "lower or equal"

[10] Most implementations allow sequence number wrap-around. For simplicity, we assume that $j > i$ implies that $TPDU_j$ is later in the data flow than $TPDU_i$.

combines good features of selective repeat and Go-Back-N by having the transport receiver return both a cumulative PACK, and a selective PACK for the TPDU that most recently arrived [Keshav and Morgan 1997]. This combined ACK information helps the transport sender avoid unnecessary retransmissions. Further work on innovative retransmission strategies for use over wireless networks can be found in [Balakrishnan et al. 1996] (see Section 6.2). Both selective repeat and Go-Back-N also apply to data link layer communication, and are thoroughly analyzed in [Stallings 1997; Tanenbaum 1996; Walrand 1991].

4.4.6 *Sender-dependent vs. Sender-independent Acknowledgment.* In some protocols, transport receiver ACKs are generated in response to explicit or implicit events initiated by the transport sender. These ACKs are called *sender-dependent* [Doeringer et al. 1990; Thai et al. 1994]. In XTP, no ACKs are generated until the transport sender explicitly instructs the transport receiver to generate one. Some transaction-oriented protocols such as VMTP try to minimize the number of control TPDUs exchanged by using the response as an implicit PACK for a transmitted request, and a new request as an implicit PACK for the previous response.

Sender-independent ACKs are those generated by a transport receiver independent of actions by the transport sender. SNR/ESNR [Doshi et al. 1993; Netravali et al. 1990] uses an error reporting and recovery scheme called *Periodic State Exchange* where complete state information is exchanged periodically between a transport sender and a transport receiver based on timers, not the arrival or loss of TPDUs. If a control TPDU containing state information is lost, no immediate steps are taken. Eventually after another period, the complete state information containing information in the lost control TPDU will be exchanged. This approach simplifies the error recovery mechanism and reduces the number of error recovery timers. With parallel processors, one processor at the receiver can be dedicated to processing state information and inserting control TPDUs while a second focuses on processing TPDUs.

4.4.7 *Duplicate Detection.* Sequence numbers are also the basic mechanism for *duplicate detection.* If an ACK is lost and as a result one or more TPDUs are retransmitted, duplicate TPDUs can arrive at the transport receiver. In most CO protocols, the transport receiver maintains state information about previously received TPDUs. (A less common approach to duplicate detection uses synchronized clocks instead of state information [Liskov et al. 1990]). When a duplicate TPDU is received, the state information allows it to be detected. In CL protocols, where no state information is maintained, duplicate detection is not possible and the typical offered service is maybe-loss, maybe-duplicates. In CL protocols that use ACKs and retransmissions, the offered service is limited to no-loss, maybe-duplicates service.

When a duplicate is received after a connection close and subsequent reconnection between the same two transport entities, duplicate detection becomes a delicate problem. In this case, the duplicate may accidentally be considered original data for the second connection. Several solutions exist for this *delayed duplicate* problem. First, each transport entity can remember the last sequence number used for each terminated connection (i.e., maintain state even after disconnection). If a connection is re-established, the next sequence number is used. A design problem involves deciding how long each transport entity must maintain the state information. A second approach uses a unique connection identifier for each connection. Both approaches work fine unless the system loses the sequence number or connection identifier information, say due to a system crash. A third approach requires a minimum amount of time before a connection between the same two transport entities can be re-established, and enforces a maximum lifetime of any TPDU to be less than that minimum. This so-called TIME-WAIT approach is used in TCP; it has the disadvantage of introducing artificial delays between consecutive connections.

4.4.8 *Forward Error Correction.* Because each retransmission requires at least one round-trip time, PAR and ARQ schemes may operate poorly for applications that have tight latency constraints. As an alternative, *Forward Error Correction* (FEC) can be used for low latency error control, with or without PAR/ARQ. In TP++, a transport sender can transmit each TSDU as k data TPDUs and an additional h redundant parity TPDUs. Unless the network loses more than h of the $h+k$ TPDUs sent, the transport receiver can reconstruct the original k data TPDUs.

4.5 Flow/Congestion Control

The terms flow control and congestion control create confusion, since different authors approach these subjects from different perspectives. In this section, we try to make a useful distinction between the terms while recognizing that overlap exists.

First, we define transport layer *flow control* as any scheme by which the transport sender limits the rate at which data is sent over the network. The goals of flow control may include one or both of the following: (1) preventing a transport sender from sending data for which there is no available buffer space at the transport receiver, or (2) preventing too much traffic in the underlying network. Flow control for (2) also is called *congestion control* or *congestion avoidance.* Congestion is essentially a network layer problem, and dozens of schemes are discussed and classified in [Yang and Reddy 1995]. However, congestion often is addressed by transport layer flow control (also known as end-point flow control).

Techniques used to implement both goals are often tightly integrated, as is the case in TCP. Regardless of which goal is being pursued, the flow of data is being controlled. Therefore, some authors use "flow control" to refer to protocol features that address both goals, thus blurring the distinction between flow and congestion control [Stevens 1994, p. 310]. Bertsekas [Bertsekas and Gallager 1992] states that flow and congestion control overlap so much that it is better to treat them as a single problem; however, he discusses flow control as a major network layer function. Other authors [Stallings 1997; Tanenbaum 1996; Walrand 1991] clearly separate congestion control from flow control. Feldmeier states that congestion control can be viewed as a generalized n-dimensional version of flow control, and suggests that flow control for all layers should be performed at the lowest layer that requires flow control [Feldmeier 1993a]. In this paper, we use flow control as the overall term for such techniques, but emphasize that (1) flow control is only one example of many possible congestion control techniques, and (2) congestion control is only one of the two motivations for flow control.

We now present a discussion of general flow control techniques, and a discussion of how these techniques combat network congestion.

4.5.1 *General Flow Control Techniques.* Transport layer flow control is more complex than network or data-link layer flow control. This is because storage and forwarding at intermediate routers causes highly variable delays that are generally longer than actual transmission time [Stallings 1997, p.593]. Transport layer flow control usually is provided in tandem with error control, by using sequence numbers and windowing techniques. Some authors, however, claim that error control and flow control have different goals and should be separated in high-speed network protocol design [Feldmeier 1990; La Porta and Schwartz 1991; Lundy and Tipici 1994].

Two techniques may be used, either alone or together, to avoid network congestion and overflowing receiver buffers. These are: (1) window flow control and (2) rate control.

In *window* (or *sliding-window*) flow control, the transport sender continues sending new data as long as there remains space in the sending window. In general, this window may be fixed or variable in size. In fixed size window control, ACKs from the transport receiver are used to advance the transport sender's window.

In variable size window control, also called a *credit scheme*, ACKs are decoupled from flow control. A TPDU may be acknowledged without granting the transport sender additional credit to send new TPDUs, and additional credit can be given without acknowledging a TPDU. The transport receiver adopts a policy concerning the amount of data it permits the transport sender to transmit, and *advertises* the size of this window in TPDUs that flow from receiver to sender.

Early experience with TCP showed a problem, known as the *Silly Window Syndrome*, that can afflict protocols using the credit scheme. The silly window syndrome can start either because the transport receiver advertises a very small window, or the transport sender sends a very small TPDU. Clark [Clark 1982] describes how this can degenerate into a vicious cycle where the average TPDU size ends up being much smaller than the optimal case, and throughput suffers as a result. TCP includes mechanisms in both the transport sender and transport receiver to avoid the conditions that lead to this problem.

Credit schemes may be described as conservative or aggressive (optimistic). The conservative approach, as used in TCP, only allows new TPDUs up to the transport receiver's available buffer space. This approach has the disadvantage that it may limit a transport connection's throughput in long delay or large bandwidth-product situations. The aggressive approach attempts to increase throughput by allowing a transport receiver to optimistically grant credit for space it does not have [Stallings 1997, p.598]. A disadvantage of the aggressive approach is its potential for buffer overflow at the transport receiver (hence, wasted bandwidth) unless the credit-granting mechanism is carefully timed [Clark 1982].

Rate control uses timers at the transport sender to limit data transmission [Cheriton 1988; Clark et al. 1987; Weaver 1994]. Either a transport sender can be assigned (1) a *burst size* and *interval* (or *burst rate*), or (2) an *interpacket delay time*. In (1), a transport sender transmits a burst of TPDUs at its maximum rate, then waits for the specified burst interval before transmitting the next burst (e.g., NETBLT and XTP). This is often modeled as a *token-bucket* scheme. In (2), a transport sender transmits data as long as it has credit available, but artificially pauses between each TPDU according to the interpacket delay (e.g., VMTP [Cheriton and Williamson

1989]). This is often modeled as a *leaky-bucket* scheme. Case (2) improves performance because spreading TPDU transmissions helps avoid network congestion and receiver overflow. However, interpacket delay algorithms are more difficult to implement because of the timers needed.

Rate control schemes involve low network overhead since they do not exchange control TPDUs except to occasionally adjust the rate in response to a significant change in the network or the transport receiver. They also do not affect the way data ACKs are managed [Doeringer et al. 1990]. Some authors argue that implementing window control and rate control together as in XTP and NETBLT will achieve better performance than implementing only one of them [Clark et al. 1987; Doeringer et al. 1990; La Porta and Schwartz 1991; Sanders and Weaver 1990].

One innovative approach implemented in TP++ involves *backpressure*, where the application, transport, and network layers all interact. If a user receiver reads from the transport receiver at a rate slower than the transport sender is sending, the transport receiver's buffers eventually will fill up. The transport receiver explicitly triggers congestion control procedures within the network. The network sender in turn refuses additional TPDUs from the transport sender. The transport sender may then exercise backpressure on the sending application by refusing to accept any more new APDUs, until notice arrives that the transport receiver's buffers have started to empty [Stevens 1994, p.398]. This method will not work if multiple transport connections are multiplexed on a single network connection [Stallings 1997, p.596].

4.5.2 *Flow Control Techniques for Addressing Congestion.* Congestion control schemes have two primary objectives: fairness, and optimality.

Fairness involves sharing resources among users (individual data flows) fairly. Since transport entities lack knowledge about general network resources, the bottleneck(s) itself is at a better position to enforce fairness. The transport layer's role is limited to ensuring that all transport senders cooperate. For example, TCP's congestion control technique assumes that all TCP entities will "back-off" in a similar manner in the presence of congestion. Unfortunately, it only works in a cooperative environment. Today's Internet is experiencing congestion difficulties because of some non-TCP-conformant transport entities acting selfishly. Modifications to *Random Early Dropping* (RED) algorithms have been proposed to counter these selfish TCP implementations [Floyd and Jacobson 1993].

Optimality involves ensuring that all routers operate at optimal levels. Transport entities have a role to play by limiting the traffic sent over any router that becomes a bottleneck. A router can monitor itself and send *explicit access control* feedback to the traffic sources [Prue and Postel 1987; Ramakrishnan and Jain 1988]. Alternatively, transport entities can use *implicit access control* and detect potential bottlenecks from timeouts and delays derived from the ACKs received [Ahn et al. 1995; Jain 1989]. A third method is to use packet-pair flow control which also has the advantage of ensuring that well-behaved users are protected from ill-behaved users because of the use of round-robin schedulers [Keshav ; Keshav 1991].

Explicit access control has been implemented in XTP and TP++. XTP implements both end-to-end flow control and explicit access congestion control. Initially, default parameters control the transport sender's transmission rate. However, during the exchange of control information, the network can modify the flow control parameters to control congestion. Furthermore, network routers can explicitly send control PDUs to transport entities to control the congestion. TP++ uses the previously mentioned backpressure both to avoid overwhelming a slow receiver and to control congestion.

When the network layer provides no support for explicit access control, the transport protocol may operate on the assumption that any TPDU loss indicates network congestion — a reasonable assumption when the underlying network links are highly reliable. TCP's approach to congestion avoidance incorporates two important design principles. The first principle is *slow-start*, where new connections do not initially send at the full available bandwidth, but rather first send one TPDU per RTT, then two, then four, etc., (exponential increase per RTT) until the full bandwidth is reached, or a loss is detected. The second principle is that when loss is encountered, indicating potential congestion, the transport sender reduces its sending rate significantly, and then uses an additive increase per RTT in an attempt to find an optimal sending rate. A goal of TCP is that network bandwidth is shared fairly when all TCP connections abide by both principles. Details of these principles have been refined over various implementations of TCP, which go by the names *Tahoe, Reno*, and *Vegas* [Ahn et al. 1995; Fall and Floyd 1996; Jacobson 1988].

Using a control-theoretic approach, congestion control schemes can be viewed as a control policy to achieve prescribed goals (e.g., minimize round-trip delay). These schemes are divided into *open loop* (e.g., rate control and implicit access control schemes), where control decisions by the transport sender do not depend on feedback from the congested sites, and *closed loop* (e.g., explicit access control schemes), where control decisions do depend on such feedback. With networks having higher speeds and higher bandwidth-delay products, open loop schemes have been proposed as being more effective. Many current admission control schemes, which can be called *regulation* schemes, are open loop.

4.6 Multiplexing/Demultiplexing

Multiplexing, as shown in Figure 5a, supports several transport layer connections using a single network layer association. Multiplexing maps several user/transport interface points (what ISO calls TSAPs) onto a single transport/network interface point (NSAP). An association is a virtual circuit in the case of a CO network. In the case of a CL network, an association is a pair of network addresses (source and destination). When the underlying network is CL (as is the case for TCP), multiplexing/demultiplexing as provided by TCP's port numbers is necessary to serve multiple transport users.

Multiplexing uses network layer resources more efficiently by reducing the network layer's context-state information. Additionally it can provide primitive stream synchronization [Feldmeier 1990]. For example, video and audio streams of a movie can be multiplexed to maintain "lip sync" in which case the multiplexed streams are handled as a single stream in a single, uniform manner. This method works only if all streams require the same network layer QoS. Multiplexing's advantages come at the expense of having to *demultiplex* TPDUs at the transport receiver and to ensure fair sharing of resources among the multiplexed connections. Demultiplexing requires a look-up operation to identify the intended receiver and possibly extra process switching and data copying.

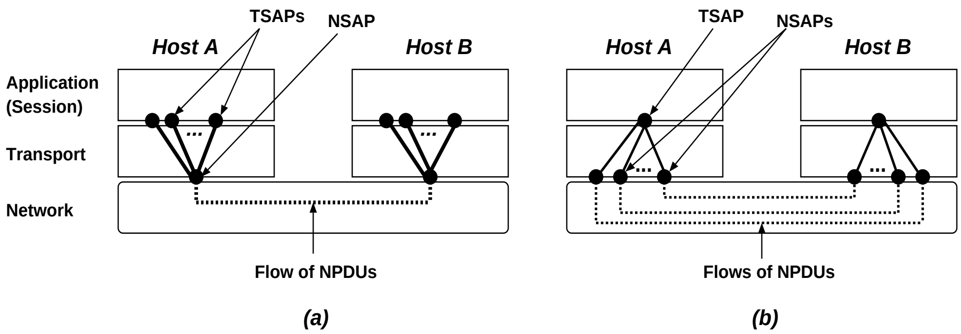

Fig. 5. (a) Multiplexing/Demultiplexing (b) Splitting/Recombining

4.7 Splitting/Recombining

Splitting (or *downward-multiplexing* [Walrand 1991]) as shown in Figure 5b is the opposite of multiplexing. In this case several network layer associations support a single transport layer connection. Splitting provides additional resilience against a network failure and potentially increases throughput by letting a fast processor output data over several slower network connections [Strayer and Weaver 1988].

4.8 Concatenation/Separation

Concatenation as shown in Figure 6 combines several TPDUs into one network service data unit (NSDU), thus reducing the number of NSDUs submitted to the network layer. The transport receiver has to *separate* the concatenated TPDUs. This mechanism saves network bandwidth at the expense of extra processing time at the transport sender and receiver. Considering that the transport receiver is often the bottleneck in protocol processing and that available bandwidth is increasing, some authors argue that concatenation/separation and splitting/recombining mechanisms are becoming less important [Thai et al. 1994].

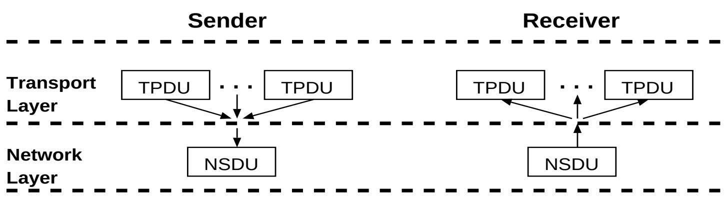

Fig. 6. Concatenation and Separation

4.9 Blocking/Unblocking

Blocking combines several TSDUs into a single TPDU (see Figure 7a) thereby reducing the number of transport layer encapsulations and the number of NSDUs submitted to the network layer. The receiving transport entity has to *unblock* or *deblock* the blocked TSDUs before delivering them to transport service user.

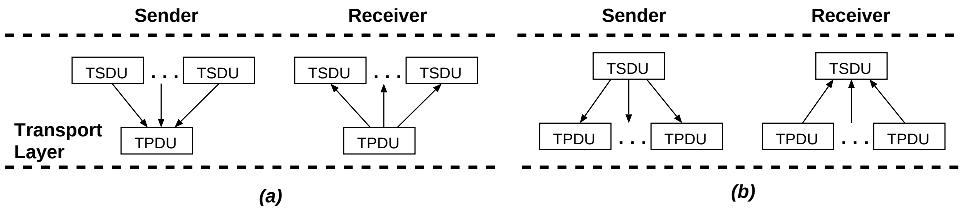

Fig. 7. (a) Blocking/Unblocking (b) Segmentation/Reassembly

Blocking/unblocking is intended to save network bandwidth. The effect on processing time varies with the particular implementation. A protocol that has to do both blocking and unblocking, say to preserve TSDU boundaries, might add processing time. However, TCP's form of blocking, described by the Nagle Algorithm [Nagle 1984], actually may reduce processing time at the transport receiver, since it reduces the number of TCP headers which must be processed. TCP's transport receiver unblocks, but does not preserve TSDU boundaries. This is consistent with TCP's CO-byte service.

Although blocking and concatenation are similar (both permit grouping of PDUs), they may serve different purposes. As one subtle difference, concatenation permits the transport layer to group control (e.g., ACK) TPDUs with data TPDUs that were derived from TSDUs. Piggybacking ACKs and data is an example of concatenation. Blocking only combines TSDUs (i.e., transport layer data). Segmentation and concatenation may occur, but blocking, as defined within the ISO Reference Model, in CL protocols is not permitted [ITU-T 1994b].

4.10 Segmentation/Reassembly

Often the network service imposes a maximum permitted NSDU size. In such cases, a larger TSDU is *segmented* into smaller TPDUs by the transport sender as shown in Figure 7b. The transport receiver *reassembles* the TSDU before delivering it to the user receiver.

TCP segmentation is based on negotiating a maximum segment size during connection establishment, or in more recent implementations, on a Path Maximum Transmission Unit (MTU) discovery mechanism. To avoid the overhead associated with network layer fragmentation, many TCP implementations determine the largest size IP datagram that can be transmitted end-to-end without IP fragmentation [Kent and Mogul 1987]. These implementations then use transport layer segmentation (and/or blocking) with this MTU in mind, thereby reducing[11] the

[11] Due to dynamic route changes, the MTU can change with TCP; thus IP fragmentation can be avoided but not prevented.

risk for network layer segmentation [Stevens 1994, p.340].

5. TRANSPORT PROTOCOL EXAMPLES

This section now summarizes eleven widely implemented or particularly influential transport protocols other than TCP. Each one's features are described in terms of the service and protocol features discussed in general in Sections 3 and 4, respectively. The protocol descriptions are summarized from the indicated references and from direct correspondence with those people most responsible for each protocol's design and development. Together these protocols are summarized in Tables 2–4.

The last part of the section provides a brief statement about eight experimental transport protocols that appear in the literature.

5.1 UDP

User Datagram Protocol (UDP) (see Table 2) provides CL, uncontrolled-loss, maybe-duplicates, unordered service [Postel 1980]. UDP is basically an interface to IP, adding little more than TSAP demultiplexing (port numbers) and optional data integrity service. By not providing reliability service, UDP's overhead is significantly less than that of TCP. Although UDP includes an optional checksum, there is no provision for error reporting; incoming TPDUs with checksum errors are discarded, valid ones are passed to the user receiver.

5.2 TP4

The ISO Transport Protocol Class 4 (or TP4) (see Table 2) was designed for the same reasons as TCP. It provides a similar CO, reliable service over a CL unreliable network. TP4 relies on the same mechanisms as TCP with the following differences. First, TP4 provides a CO-message service rather than CO-byte. Therefore, sequence numbers enumerate TPDUs rather than bytes. Next, sequence numbers are not initiated from a clock counter as in TCP [Stevens 1994], but rather start from 0. A destination reference number is used to distinguish between connections. This number is similar to the destination port number in TCP, but here the reference number maps onto the port number and can be chosen randomly or sequentially [Bertsekas and Gallager 1992]. Another important difference is that (at least in theory) a set of QoS parameters (see Section 3.8) can be negotiated for a TP4 connection. Other differences between TCP and TP4 are discussed in [Piscitello and Chapin 1993].

5.3 TP0

The ISO Transport Protocol Class 0 (or TP0) (see Table 2) was designed as a minimum transport protocol providing only those functions necessary to establish a connection, transfer data, and report protocol errors. TP0 was designed to operate on top of a CO reliable network service that also provides end-to-end flow control. TP0 does not even provide its own disconnection procedures; when the underlying network connection closes, TP0 closes with it. One interesting use of TP0 is that it can be employed to create an OSI transport service on TCP's reliable byte-stream service, enabling OSI applications to run over TCP/IP networks [Rose and Cass 1987].

5.4 NETBLT

Network Block Transfer (NETBLT) (see Table 3) was developed at MIT for high throughput bulk data transfer [Clark et al. 1987]. It is optimized to operate efficiently over long-delay links. NETBLT was designed originally to operate on top of IP, but can operate on top of any network protocol that provides a similar CL unreliable network service. Data exchange is realized via unidirectional connections. The unit of transmission is a buffer, several of which can be concurrently active to keep data flowing at a constant rate. Connection is established via a 2-way-handshake during which buffer, TPDU and burst sizes are negotiated. Flow control is accomplished using buffers (transport-user-level control) and rate control (transport-protocol-level control). Either transport user of a connection can limit the flow of data by not providing a buffer. Additionally, NETBLT uses burst size and burst rate parameters to accomplish rate control. NETBLT uses selective retransmission for error recovery. After a transport sender has transmitted a whole buffer, it waits for a control TPDU from the transport receiver. This TPDU can be a *RESEND*, indicating lost TPDUs, or an *OK*, acknowledging the whole buffer. A *GO* allows the transmission of another buffer. Instead of waiting after each buffer, a multiple buffering mechanism can be used [Dupuy et al. 1992].

5.5 VMTP

The Versatile Message Transaction Protocol (VMTP) (see Table 3) was designed at Stanford University to provide high performance communication service for distributed operating systems, including file access, remote procedure calls (RPC), real-time datagrams and multicast [Cheriton and Williamson 1989]. VMTP is a request-response protocol that uses timer-based connection management to provide communication between network-visible entities. Each entity has a 64-bit identifier that is unique, stable, and independent of host-address. The latter property allows entities to be migrated and handled independent of network layer addressing, facilitating process migration and mobile and multi-homed hosts [Cheriton and Williamson 1989]. Each request (and response) is identified by a transaction identifier. In the common case, a client increments its transaction identifier and sends a request to a single server; the server sends back a response with the same (Client, Transaction) identifier. A response *implicitly* acknowledges the request and each new request *implicitly* acknowledges the last response sent to this client by the server. Multicast is realized by sending to a group of servers. Datagram support is provided by indicating in the request that no response is expected. Additionally, VMTP provides a streaming mode in which an entity can issue a stream of requests, receiving the responses back asynchronously [Williamson and Cheriton 1989]. Flow control is achieved by a rate control scheme borrowed from NETBLT with negotiated interpacket delay time.

5.6 T/TCP

Transaction TCP (T/TCP) (see Table 3) is a backwards-compatible extension of TCP that provides efficient transaction-oriented service in addition to CO service [Braden 1992a; Braden 1994]. The goal of T/TCP is to allow each transaction to be efficiently performed as a single incarnation of a TCP connection. It introduces two major improvements over TCP. First, after an initial transaction is handled using a 3-way-handshake connection, subsequent transactions streamline connection establishment through the use of a 32-bit incarnation number, called a "connection count" (CC) carried in each TPDU. T/TCP uses the monotonically increasing CC values in initial *CR-TPDU*s to bypass the 3-way-handshake, using a mechanism called *TCP Accelerated Open*. With this mechanism, a transport entity needs to cache a small amount of state for each remote peer entity. The second improvement is that T/TCP shortens the delay in the TIME-WAIT[12] state. T/TCP defines three new TCP options, each of which carries one 32-bit CC value. These options accelerate connection setup for transactions. T/TCP includes all normal TCP semantics, and operates exactly as TCP for all features other than connection establishment and termination.

5.7 RTP

Real-time Transport Protocol (RTP) (see Table 3) was designed for real-time multi-participant multimedia applications [Schulzrinne 1996; Schulzrinne et al. 1996]. Even though RTP is called a transport protocol by its designers, this sometimes creates confusion, because RTP by itself does not provide a complete transport service. RTP TPDUs must be encapsulated within the TPDUs of another transport protocol that provides framing, checksums, and end-to-end delivery, such as UDP. The main transport layer functions performed by RTP are to provide timestamps and sequence numbers for TSDUs. These timestamps and sequence numbers *may* be used by an *application* written on top of RTP to provide error detection, resequencing of out-of-order data, and/or error recovery. However it should be emphasized that RTP itself does not provide any form of error detection or error control. Furthermore, in addition to transport layer functions, RTP also incorporates presentation layer functions; through the use of so-called *RTP profiles*, RTP provides a means for the application to identify the format of data (i.e., whether the data is audio or video, what compression method is used, etc.).

RTP has no notion of a connection; it may operate over either CO or CL service. Framing and segmentation must be done by the underlying transport layer. RTP is typically implemented as part of the application and not in the operating system kernel; it is a good example of Application Level Framing and Integrated Layer Processing [Clark and Tennenhouse 1990]. It consists of two parts: data and control. Continuous media data such as audio and video is carried in RTP data TPDUs. Control information is carried in RTCP (RTP Control Protocol) TPDUs. Control TPDUs are multicast periodically to the same multicast group as data TPDUs. The functions of RTCP can be summarized as QoS monitoring, inter-media synchronization, identification, and session size estimation/scaling. The control traffic load is scaled with the data traffic load so that it makes up a certain percentage of the data rate (5%).

RTP and RTCP were designed under the auspices of the Internet Engineering

[12]When a TCP entity performs an active close and sends the final ACK, that entity must remain in the TIME-WAIT state for twice the *Maximum Segment Lifetime* (MSL), the maximum time any TPDU can exist in the network before being discarded. This allows time for the other TCP entity to send and resend its final ACK in case the first copy is lost. This closing scheme prevents TPDUs of a closed connection from appearing in a subsequent connection between the same pair of TCP entities [Braden 1992b].

Task Force (IETF).

5.8 APPN (SNA)

The first version of IBM's proprietary Systems Network Architecture (SNA) was implemented in 1974 in a hierarchical, host-centric manner: the intelligence and control resided at the host, which was connected to dumb terminals. The growing popularity of LAN-based systems led to SNA version 2, a dynamic peer-to-peer architecture called *Advanced Peer-to-Peer Networking* (APPN) [Chiong 1996; Dorling et al. 1997]. Version 3, called *High Performance Routing* (HPR), is a small but powerful extension to APPN [Dorling et al. 1997; Freeman 1995]. HPR includes APPN-based high-performance routing, which addresses emerging high-speed digital transmissions. Although SNA is a proprietary architecture, its structure has moved closer to TCP/IP with each revision. Today, SNA exists mostly for legacy reasons. SNA and TCP/IP are the most widely used protocols in the enterprise networking arena [Chiong 1996, p.31].

APPN does not map nicely onto ISO's modular layering since it does not define an explicit transport protocol per se. However, end-to-end transport layer functions such as end-to-end connection management are performed as part of APPN. Its CO service is built on top of a reliable network service. Therefore, many functions related to error control that are commonly handled at the transport layer are not required in APPN's transport layer. Unlike applications using TCP, SNA applications do not directly interface to the transport layer. Instead applications are based on a "Logical Unit" concept which contains the services of all three layers above network layer. An interesting feature of APPN is its Class of Service(COS)-based route selection where the route can be chosen based on the associated cost factor. The overall cost for each route depends on various factors such as the speed of the link, cost per connection, cost per transaction, propagation delay, and other user-configurable items such as security. For service and protocol details of APPN, see Table 4.

HPR, an extension to APPN, enhances data routing performance by decreasing intermediate router processing. The main components of HPR are *Automatic Network Routing* (ANR) and *Rapid Transport Protocol* (RTP) [Chiong 1996; Dorling et al. 1997]. Unlike APPN, RTP is an explicit transport protocol which provides a CO-message, reliable service. Some of the key features of RTP include bidirectional data transfer, end-to-end connections without intermediate session switching, end-to-end flow and congestion control based on adaptive rate-based (ARB) control, and automatic switching of sessions without service disruption.

5.9 NSP (DECnet)

DECnet is a proprietary protocol suite from Digital Equipment Corporation. Early versions (1974-1982) were limited in functions, providing only point-to-point links, or networks of ≤ 255 processors. The first version capable of building networks of thousands of nodes was DECnet Phase IV in 1982 (two years earlier than the OSI Reference Model was standardized). Its layer corresponding most closely to the OSI Transport layer was called the "End Communication Layer"; it consisted of a proprietary protocol called the *Network Services Protocol* (NSP).

NSP provides CO-message and reliable service. It supports segmentation and reassembly, and has provisions for flow control and congestion control [Robertson 1996, p.50ff]. Two data channels are provided: a "Normal-Data" channel and an "Other-Data" channel which carries expedited data and messages related to flow control. DEC engineers claim that OSI's TP4 "is essentially an enhancement and refinement of Digital's proprietary NSP protocol" [Martin and Leben 1992]. Indeed, there are many similarities between TP4 and NSP, such as their mechanisms for handling expedited data.

In the early 1990s, DECnet Phase V was introduced to integrate OSI protocols with the earlier proprietary protocols. Phase V supports a variant of TP4, plus TP0 and TP2 and NSP for backwards compatibility. A session level service known as "tower matching" resolves which protocol(s) are available for a particular connection, and selects a specific protocol at connection establishment.

DECnet was widely used throughout the 1980's, and remains an important legacy protocol for many organizations. DECnet was one of the two target protocol suites for the original implementation of the popular X11 windowing system (the other being TCP/IP).

5.10 XTP

The Xpress Transport Protocol's design (XTP Version 4.0[13]) (see Table 4) was coordinated within the XTP Forum to support a variety of applications such as multimedia distribution and distributed applications over WANs as well as LANs [Strayer et al. 1992]. Originally XTP was designed to be implemented in VLSI; hence it has a 64-bit alignment, a fixed-size header, and fields likely to control a TPDU's initial processing located early in the header. However, no hardware implementations were ever built. XTP combines classic functions of TCP, UDP, and TP4, and adds new services such as transport multicast, multicast group management, priorities, rate and burst control, and selectable error and flow control mechanisms. XTP can operate on top of network protocols such as IP or ISO CLNP, data link protocols such as 802.2, or directly on top of the AAL of ATM. XTP simply requires framing and end-to-end delivery from the underlying service. One of XTP's most important features is the orthogonality it provides between communication paradigm, error control and flow control. An XTP user can choose any communication paradigm (CO, CL, or transaction-oriented) and whether or not to enable error control and/or flow control. XTP uses both window-based and rate-based flow control.

5.11 SSCOP/AAL5 (ATM)

In the ATM environment, several transport protocols have been and are being developed; they are referred to as ATM Adaptation Layers (AALs).[14] These AALs have been specified over time to handle different traffic classes [Stallings 1998, Section 4.4] - CO constant bit rate (CBR) requiring synchronization, CO variable

[13] XTP version 3.6 (the Xpress Transfer Protocol) was a transport and network layer protocol combined. XTP 4.0 performs only transport layer functions.

[14] Some authors note that since none of the AALs provide reliable service, then "It is not really clear whether or not ATM has a transport layer." [Tanenbaum 1996, p.545]. We argue that reliability or the lack thereof is not the issue; since ATM virtual circuits/paths are a concatenation of links between store-and-forward NPDU (cell) switches, then by definition, the AAL protocols above ATM provide end-to-end transport service.

bit rate (VBR) requiring synchronization, CO-VBR not needing synchronization, CL-VBR not requiring synchronization, and most recently available bit rate (ABR) where bandwidth and timing requirements are defined by the user in an environment where an ATM backbone provides the same quality of service as found in a LAN [Black 1995].

Although 5 transport protocols (AAL1-5) have been proposed for these traffic classes, AAL5 which supports virtually all data applications has greatest potential for marketplace success. None of the AALs including AAL5 provide reliable service (although with minor changes AAL5 could).[15] Another protocol that does provide end-to-end reliability is the Service Specific Connection-Oriented Protocol (SSCOP) [ITU-T 1994a] which can run on top of AAL5.

SSCOP was initially standardized to provide for reliable communication of control signals [ITU-T 1995b], not data transfer. It has since been approved for reliable data transfer with I.365.3 [ITU-T 1995a] specifying it as the basis for providing ISO's connection-oriented transport service [ITU-T 1995c]. Some authors cite SSCOP as *potentially* applicable as a general purpose end-to-end reliable transport protocol in the ATM environment [Henderson 1995]. For these reasons, we select SSCOP/AAL5 as the ATM transport protocol to survey in Table 4.

SSCOP incorporates a number of important design principles of other high-speed transport protocols (e.g., SNR [Lundy and Tipici 1994]) including the use of complete state exchange between transport sender and receiver to reduce reliance on timers. SSCOP's main drawback for general usage is that it is designed to run over an underlying ATM service that provides in-order PDU delivery. The SSCOP receiver assumes that every out-of-order PDU indicates a loss (as opposed to possible misordering) and immediately returns a NACK (called a USTAT) which acts as a selective reject (see Section 4.4.2). Over an unordered service, SSCOP would still correctly provide its advertised reliable service; it is simply unclear whether SSCOP would be efficient in an environment where the underlying service misorders PDUs. One author indicates that SSCOP could be modified to be a more general robust transport protocol capable of running above an unreliable, connectionless service [Henderson 1995].

5.12 Miscellaneous Transport Protocols

The following are brief descriptions of eight experimental transport protocols that influenced transport protocol development. While many of the ideas contained in these protocols are innovative and useful, for one reason or another, they have not themselves been successful in the marketplace. This may be the result of: (1) the marketing problem of introducing new transport protocols into an existing infrastructure, (2) the primarily university nature of the protocol without added industrial support, or (3) the failure of an infrastructure for which the protocol has been specifically designed.

Although these protocols were surveyed as were TCP and the previous eleven,

[15] Because of AAL's lack of reliable service and the popularity of TCP/IP, there exists a protocol stack with TCP over IP over AAL over ATM. This stack essentially puts a transport layer over a network layer over a transport layer over a network layer to provide reliable data transfer in a mixed TCP/IP - ATM environment [Cole et al. 1996].

space limitations prevent presenting them in detail. The tables that summarize these protocols are available at: `www.cis.udel.edu/~amer/PEL/survey/`.

Delta-t was designed in the late 1970s for reliable stream and transaction-oriented communications on top of a best effort network such as IP or ISO CLNP [Watson 1989]. Its main contribution is in connection management: it achieves hazard-free connection management without exchanging explicit control TPDUs.

SNR was designed in the late 1980s for a high speed data communications network [Doshi et al. 1993; Lundy and Tipici 1994]. Its key ideas are periodic exchange of state information between the transport sender and the transport receiver, reduction of protocol overhead by making decisions about blocks of TPDUs instead of individual ones (*packet blocking*), and parallel implementation of the protocol on a dedicated front-end communications processor. SNR uses three modes of operation: one for virtual networks, one for real-time applications over reliable networks, and one for large file transfers.

The *MultiStream Protocol* (MSP), a feature-rich, highly flexible CO transport protocol designed in the early 1990s, supports applications that require different types of service for different portions of their traffic (e.g., multimedia). Transport users can dynamically change modes of operation during the life of a connection without loss of data [Porta and Schwartz 1993a; Porta and Schwartz 1993b]. MSP defines seven traffic streams, each of which is defined by a set of common protocol functions. These functions specify the order in which TPDUs will be accepted and passed to the user receiver.

TP++ was developed in the early 1990s. It is intended for a heterogeneous internetwork with a large bandwidth-delay product [Biersack et al. 1992; Feldmeier 1993b]. It uses backpressure for flow and congestion control, and is designed to carry three major application classes: constrained latency service, transactions, and bulk data transfer.

DTP, a slightly modified version of TCP, was developed in the early 1990s [Sanghi and Agrawala 1993]. Some key features that distinguish DTP from TCP are its use of a send-time control scheme for flow control, selective as well as cumulative PACKs, and TPDU-based sequence numbers.

Partial Order Connection (POC) was recently designed to offer a middle ground between the ordered/no-loss service provided by TCP, and the unordered/uncontrolled-loss service provided by UDP [Amer et al. 1994]. The main innovation of POC is the introduction of *partially-ordered/controlled-loss service* [Conrad ; Conrad et al. 1996; Marasli 1997; Marasli et al. 1997].

Two recent transport protocols, the *k-Transmit Protocol* (k-XP) and the *Timed-Reliable Unordered Message Protocol* (TRUMP), provide CO-message, unordered, controlled-loss service. k-XP was implemented as an application software library that provides several major enhancements to the basic service provided by UDP, such as connection management and controlled-loss delivery [Amer et al. 1997]. TRUMP is a variation of k-XP that allows applications to specify an expiration time for each TSDU [Golden 1997].

6. FUTURE DIRECTIONS AND CONCLUSION

In this section, we first summarize how several recent trends and technological developments have impacted transport layer design. We then cover a particular trend in more detail; namely, the impact of wireless networking on the transport layer. Next, we enumerate some of the debates concerning transport layer design. We conclude with a few final observations.

6.1 Impacts of Trends and New Technologies

Several trends have influenced the design of transport protocols over the last two decades.

Faster satellite links and *gigabit networks* have resulted in networks with larger end-to-end bandwidth-delay products. These so-called "Long Fat Networks" allow increased amounts of data in transit at any given moment. This required extensions to TCP, for example, to prevent wrapping of sequence numbers [Borman et al. 1992].

Fiber optics has improved the quality of communication links, thus shifting the major source of network errors—at least for wired networks—from line bit errors to NPDU losses due to congestion. This fact is exploited by TCP's Congestion Avoidance and Fast Retransmit and Recovery algorithms which were made necessary because of the *exponential growth* in the use of the Internet—even before the Web made the Internet a household word. Congestion avoidance continues to be an active research area for the TCP community [Brakmo et al. 1994; Jacobson 1988; Stevens 1997].

Higher speed links and fiber optics have caused some designers of so-called *lightweight* transport protocols to shift their design goals from minimizing transmission costs (at the expense of processing power) to minimizing processing requirements (at the expense of transmission bandwidth) [Doeringer et al. 1990]. On the other hand, the proliferation of relatively low speed point-to-point (PPP) and *wireless* links has resulted in a varying and complex interconnection of low, medium, and high speed links with varying degrees of loss. This has introduced a new complication into the design of flow control and error control algorithms.

New applications such as transaction processing, audio/video transmission, and the Web have resulted in new and widely varying network service demands.

Over the years, TCP has been optimized for a particular mix of applications: that is, bulk transfer (e.g., File Transfer Protocol (FTP) and Simple Mail Transfer Protocol (SMTP)) and remote terminal traffic (e.g., telnet and rlogin). The introduction of the Web has created new performance challenges, as the request/response nature of Web interactions is a poor match for TCP's byte-stream [Heidemann 1997]. Even before the introduction of the Web, interest in request/response (transaction) protocols was reflected in the development of VMTP and T/TCP.

The delay and jitter sensitivity of audio and video based applications required transport protocols providing something other than reliable service. This has led to increased usage of UDP, which lacks congestion control features, resulting in increased Internet congestion. This has created a need for protocols that incorporate TCP-compatible congestion control, yet provide services appropriate for audio/video transmission (e.g., maybe-loss or controlled-loss). Audio/video transmission also required synchronization as enabled by RTP timestamps.

Finally, *distributed* applications involving one-to-many, many-to-one, and many-to-many communication have required a new multicast paradigm of communication. Unicast protocols such as XTP, VMTP, TP++, and UDP have been extended to support multicast.

6.2 Wireless Networks

Regarding the future of the transport layer, we note the influence of the rapid growth of wireless networks. Although transport protocols are supposed to be independent of the underlying networking technology, in practice, TCP is tailored to perform in networks where links are tethered. As more wireless links are deployed, it becomes more likely that the path between transport sender and transport receiver will not be fully wired. Designing a transport protocol that performs well over a heterogeneous network is difficult because wireless networks have two inherent differences that require, if not their own specific transport protocols, at least specific variations of existing ones.

The first difference is the major cause for TPDU gaps at the transport receiver. In wired networks missing TPDUs are primarily due to network congestion as routers discard NPDUs. In wireless networks, gaps are most likely due to bit errors and hand-off problems. On detecting a gap, TCP's transport receiver responds by invoking congestion control and avoidance algorithms to slow the transport sender and thereby reduce the network load. For wireless networks, a better response is to retransmit lost TPDUs as soon as possible [Balakrishnan et al. 1996].

Recent studies have concentrated on alleviating the effects of non-congestion-related losses on TCP performance over wireless and other high-loss networks [Bakre and Badrinath 1997; Balakrishnan et al. 1995; Yavatkar and Bhagwat 1994]. These studies follow two fundamentally different approaches. One approach hides any non-congestion-related loss from the transport sender by making the lossy link appear as a higher quality link with reduced effective bandwidth. As a result, the losses that are seen by the transport sender *are* mostly due to congestion. Some examples include: (1) reliable link layer protocols (AIRMAIL [Ayanoglu et al. 1995]), (2) splitting a transport connection into two connections (I-TCP [Bakre and Badrinath 1997]), and (3) TCP-aware link layer schemes (snoop [Balakrishnan et al. 1995]). The second approach makes the transport sender aware that wireless hops exist and does not invoke congestion control for TPDU losses. Selective ACK proposals such as TCP SACK [Mathis et al. 1996] and SMART [Keshav and Morgan 1997] can be considered examples of this approach. One study shows that a reliable link layer protocol with some knowledge of TCP provides 10-30% higher throughput than a link layer protocol without that knowledge [Balakrishnan et al. 1996]. Furthermore, selective ACKs and explicit loss notifications result in significant performance improvements.

The second inherent difference is that wireless devices are constrained in both their computing and communication power due to limited power supply. Therefore, the transport protocols used on these devices should be less complex to allow efficient battery usage. This approach is taken in Mobile-TCP [Haas and Agrawal 1997]. It uses the same splitting approach used in I-TCP, however, instead of identical transport entities on both ends of the wireless link, Mobile-TCP employs an asymmetrically-based protocol design which reduces the computation and communication overhead on the mobile host's transport entity. Because the wireless segment is known to be a single-hop connection, several transport functions can be either simplified or eliminated.

6.3 Debates

In reviewing the last two decades of transport protocol research, several things can be noted. We first note there seem to have been two schools of thought on the direction of transport protocol design: (1) the "hardware-oriented" school, and (2) the "application-oriented" school.

The hardware-oriented school feels that transport protocols should be separated from the operating system as much as possible, and implementations should be moved into VLSI, parallel, or special purpose processors [Feldmeier 1993c; Haas 1990; La Porta and Schwartz 1991; Strayer and Weaver 1988]. This school claims that the heavy usage of timers, interrupts, and memory read/writes degrades the performance of a transport protocol, and thus special purpose architectures are necessary. They would also point out that as network rates reach the Gigabit/sec range, it will be more difficult to process TPDUs in real time. Therefore TPDU formats should be carefully chosen to allow parallel processing and to avoid TPDU field limitations for sequence numbering, window size, etc. This school of thought is reflected in the design of protocols such as MSP and XTP.

By contrast, the application-oriented school prefers moving some transport layer functions out of the operating system into the user application, thereby integrating the upper layer end-to-end processing, and achieving a faster application processing pipeline. This school would claim that the bottleneck is actually in operations such as checksums and presentation layer conversion, all of which can be done more efficiently if they are integrated with the copying of data into application user space. This school of thought is reflected in the concepts of Application Level Framing (ALF) and Integrated Layer Processing (ILP) [Clark and Tennenhouse 1990]. ALF states that "the application should break the data into suitable aggregates, and the lower layers should preserve these frame boundaries as they process the data". ILP is an implementation concept which "permits the implementor the option of performing all (data) manipulation steps in one or two integrated processing loops". Ongoing research shows that performance gains can be obtained by using ALF and ILP [Ahlgren et al. 1996; Braun 1995; Braun and Diot 1995; Braun and Diot 1996; Diot and Gagnon ; Diot et al. 1995]. ALF concepts are reflected in the design of protocols such as RTP, POC, and TRUMP.

A second debate, orthogonal to the debate about where transport services should be implemented, is the debate about functionality. Older transport services tend to focus on a single type of service. Some recent protocol designers advocate high degrees of flexibility, orthogonality, and user-configurability to meet the requirements of various applications. These designers feel that: (1) a single transport protocol offer different communication paradigms, and different levels of reliability and flow control, and the transport user should be able to choose among them, and (2) protocol design should include multiple functions to support multicast, real-time data, synchronization, security, user-defined QoS, etc., that will meet the requirements of today's more complex user applications. This is reflected in the designs of XTP, MSP and POC, for example.

A third debate concerns protocol optimization. The light-weight protocol school of thought argues that since high-speed networks offer extremely low error rates, protocols should be "success-oriented" and optimized for maximum throughput [Doeringer et al. 1990; Haas 1990; La Porta and Schwartz 1991]. Others emphasize the increased deployment of wireless networks, which are more prone to bit errors; they argue that protocols should be designed to operate in both environments efficiently. The SMART retransmission strategy [Keshav and Morgan 1997] is a good example of the latter approach.

6.4 Final Observations

Several of the experimental protocols surveyed in this paper present beneficial ideas for the overall improvement of transport protocols. However, most of these schemes are not widely used. This may have more to do with the difficulty of introducing new transport protocols than with the merits of the ideas themselves.

In the 1980's, the primary battle was between TCP and ISO TP0/4—or, more broadly, between the Internet (TCP/IP) suite and OSI suite in general. Never-ending hallway discussions debated which would win. Today the victor is TCP/IP, due to, as Tanenbaum describes, a combination of "timing", "technology", "implementations," and "politics" [Tanenbaum 1996]. Sometimes the usefulness of transport protocol research is questioned—even research to improve TCP—because the market base of the current TCP version is so large that inertia prevents new good ideas from propagating into the user community. However, the successful incorporation of SACK [Mathis et al. 1996], TCP over high-speed networks [Borman et al. 1992], and T/TCP [Braden 1994] into implementations clearly shows that transport research is useful. What should be clear is that for any new idea to have influence in practice in the short term, it must be interoperable with elements of the existing TCP/IP protocol suite.

ACKNOWLEDGMENTS

The authors wish to thank several individuals: R. Marasli and E. Golden who contributed to an early draft of this paper; a number of protocol experts who reviewed individual service/protocol summaries and their respective table entries including A. Agrawala, A. Bhargava, R. Case, L. Chapin, B. Dempsey, D. Feldmeier, T. La Porta, D. Piscitello, K. Sabnani, H. Schulzrinne, D. Sanghi, T. Strayer, E. Tremblay, R. Watson, A. Weaver, C. Williamson; M. Taube and the anonymous reviewers who made valuable suggestions.

REFERENCES

Ahlgren, B., Bjorkman, M., and Gunningberg, P. 1996. Integrated layer processing can be hazardous to your performance. In W. Dabbous and C. Diot Eds., *Protocols for High-Speed Networks, V* (France, Oct. 1996), pp. 167–181. IFIP: Chapman & Hall.

Ahn, J., Danzig, P., Liu, Z., and Yan, L. 1995. Evaluation of TCP Vegas: Emulation and experiment. In *ACM SIGCOMM '95* (Cambridge, MA, Aug. 1995).

Amer, P., Chassot, C., Connolly, T., Diaz, M., and Conrad, P. 1994. Partial order transport service for multimedia and other applications. *IEEE/ACM Trans on Networking 2*, 5 (Oct.), 440–456.

Amer, P., Conrad, P., Golden, E., Iren, S., and Caro, A. 1997. Partially ordered, partially reliable transport service for multimedia applications. In *Advanced Telecommunications/Information Distribution Research Program* (College Park, MD, Jan. 1997). `www.cis.udel.edu/~amer/PEL/poc/postscript/atirp97.ps`.

Armstrong, S., Freier, A., and Marzullo, K. 1992. Multicast transport protocol. RFC 1301 (Feb.).

AYANOGLU, E., PAUL, S., LAPORTA, T., SABNANI, K., AND GITLIN, R. 1995. AIRMAIL: a link-layer protocol for wireless networks. *ACM Wireless Networks 1*, 47–60.

BAE, J., SUDA, T., AND WATANABE, N. 1991. Evaluation of the effects of protocol processing overhead in error recovery schemes for a high-speed packet switched network: link-by-link versus edge-to-edge schemes. *IEEE Journal of Selected Areas in Communications 9*, 9 (Dec.), 1496–1509.

BAKRE, A. AND BADRINATH, B. 1997. Implementation and performance evaluation of Indirect TCP. *IEEE Trans. on Computers 46*, 3 (March).

BALAKRISHNAN, H., PADMANABHAN, V., SESHAN, S., AND KATZ, R. 1996. A comparison of mechanisms for improving TCP performance over wireless links. In *ACM SIGCOMM '96* (Stanford, CA, Aug. 1996).

BALAKRISHNAN, H., SESHAN, S., AND KATZ, R. 1995. Improving reliable transport and handoff performance in cellular wireless networks. *ACM Wireless Networks 1*, 4 (Dec.), 469–481.

BERNERS-LEE, T., FIELDING, R., AND FRYSTYK, H. 1996. Hypertext transfer protocol – HTTP/1.0. RFC 1945 (May).

BERTSEKAS, D. AND GALLAGER, R. 1992. *Data Networks* (2nd ed.). Prentice-Hall, Upper Saddle River, NJ.

BHARGAVA, A., KUROSE, J., TOWSLEY, D., AND VAN LAMPORT, G. 1988. Performance comparison of error control schemes in high speed computer communication networks. In *IEEE INFOCOM* (New Orleans, April 1988).

BIERSACK, E., COTTON, C., FELDMEIER, D., MCAULEY, A., AND SINCOSKIE, W. 1992. Gigabit networking research at bellcore. *IEEE Network 6*, 2 (March), 42–48.

BLACK, U. 1995. *ATM: Foundation for Broadband Networks*. Prentice-Hall, New Jersey.

BORMAN, D., BRADEN, B., AND JACOBSON, V. 1992. TCP extensions for high performance. RFC 1323 (May).

BORMANN, C., OTT, J., GEHRCKE, H., KERSCHAT, T., AND SEIFERT, N. 1994. MTP-2: towards achieving the s.e.r.o. properties for multicast transport. In *International Conference on Computer Communications Networks* (San Francisco, CA, Sept. 1994).

BRADEN, R. 1992a. Extending TCP for transactions – concepts. RFC 1379 (Nov.).

BRADEN, R. 1992b. TIME-WAIT assassination hazards in TCP. RFC 1337 (May).

BRADEN, R. 1994. T/TCP – TCP extensions for transactions functional specification. RFC 1644 (July).

BRADEN, R. AND JACOBSON, V. 1988. TCP extensions for long-delay paths. RFC 1072 (Oct.).

BRADEN, R., ZHANG, L., BERSON, S., HERZOG, S., AND JAMIN, S. 1997. Resource reservation protocol (RSVP) – version 1 functional specification. RFC 2205 (Sept.).

BRAKMO, L., O'MALLEY, S., AND PETERSON, L. 1994. TCP Vegas: New techniques for congestion detection and avoidance. In *ACM SIGCOMM '94* (May 1994), pp. 24–35. `ftp://ftp.cs.arizona.edu/xkernel/Papers/vegas.ps`.

BRAUDES, R. AND ZABELE, S. 1993. Requirements for multicast protocols. RFC 1458 (May).

BRAUN, T. 1995. Limitations and implementation experiences of integrated layer processing. In *GISI 95* (Springer-Verlag, 1995), pp. 149–156.

BRAUN, T. AND DIOT, C. 1995. Protocol implementation using integrated layer processing. In *SIGCOMM '95* (Cambridge, MA, Sept. 1995). ACM.

BRAUN, T. AND DIOT, C. 1996. Automated code generation for integrated layer processing. In W. DABBOUS AND C. DIOT Eds., *Protocols for High-Speed Networks, V* (France, Oct. 1996), pp. 182–197. IFIP: Chapman & Hall.

BROWN, G., GOUDA, M., AND MILLER, R. 1989. Block acknowledgement: redesigning the window protocol. In *ACM SIGCOMM '89* (Austin, TX, Sept. 1989), pp. 128–135.

CHERITON, D. 1988. VMTP: versatile message transaction protocol: Protocol specification. RFC 1045 (Feb.).

CHERITON, D. AND WILLIAMSON, C. 1989. VMTP as the transport layer for high performance distributed systems. *IEEE Communications Magazine 27*, 6 (June), 37–44.

CHIONG, J. 1996. *SNA Interconnections.* McGraw-Hill.

CLARK, D. 1982. Window and acknowlegement strategy in TCP. 813 (July).

CLARK, D., LAMBERT, M., AND ZHANG, L. 1987. NETBLT: a bulk data transfer protocol. RFC 998 (March).

CLARK, D. AND TENNENHOUSE, D. 1990. Architectural considerations for a new generation of protocols. In *ACM SIGCOMM '90* (Philadelphia, PA, Sept. 1990), pp. 200–208.

COLE, R., SHUR, D., AND VILLAMIZAR, C. 1996. IP over ATM: A framework document. RFC 1932 (April).

CONNOLLY, T., AMER, P., AND CONRAD, P. 1994. An extension to TCP: Partial order service. RFC 1693 (Nov.).

CONRAD, P. Order, reliability, and synchronization in transport layer protocols for multimedia document retrieval. PhD Dissertation, CIS Dept. University of Delaware, (in progress).

CONRAD, P., GOLDEN, E., AMER, P., AND MARASLI, R. 1996. A multimedia document retrieval system using partially-ordered/partially-reliable transport service. In *Multimedia Computing and Networking 1996* (San Jose, CA, Jan. 1996).

DABBOUS, W. AND DIOT, C. 1995. High performance protocol architecture. In *IFIP Performance of Computer Networks Conference (PCN '95)* (Istanbul, Turkey, Oct. 1995).

DEERING, S. 1989. Host extensions for IP multicasting. RFC 1112 (Aug.).

DIOT, C. AND GAGNON, F. Impact of out-of-sequence processing on data transmission performance. *Computer Networks and ISDN Systems.* (To appear) `ftp://www.inria.fr/rodeo/diot/rr-oos.ps.gz`.

DIOT, C., HUITEMA, C., AND TURLETTI, T. 1995. Multimedia applications should be adaptive. In *HPCS Workshop* (Mystic (CN), Aug. 1995). IFIP.

DOERINGER, W., DYKEMAN, D., KAISERWERTH, M., MEISTER, B., RUDIN, H., AND WILLIAMSON, R. 1990. A survey of lightweight transport protocols for high speed networks. *IEEE Transactions on Communications 38*, 11 (Nov.), 2025–2039.

DORLING, B., LENHARD, P., LENNON, P., AND USKOKOVIC, V. 1997. *Inside APPN and HPR: The Essential Guide to the New SNA.* Prentice Hall.

DOSHI, B., JOHRI, P., NETRAVALI, A., AND SABNANI, K. 1993. Error and flow control performance of a high speed protocol. *IEEE Transactions on Communications 41*, 5 (May), 10 pages.

DUPUY, S., TAWBI, W., AND HORLAIT, E. 1992. Protocols for high-speed multimedia communications networks. *Computer Communications 15*, 6 (July/August), 349–358.

FALL, K. AND FLOYD, S. 1996. Simulation-based comparisons of tahoe, reno, and SACK TCP. *Computer Communication Review 26*, 3 (July), 5–21.

FELDMEIER, D. 1990. Multiplexing issues in communication system design. In *ACM SIGCOMM '90* (Philadelphia, PA, September 1990), pp. 209–219.

FELDMEIER, D. 1993a. A framework of architectural concepts for high-speed communications systems. *IEEE Journal of Selected Areas in Communications 11*, 4 (May), 480–488.

FELDMEIER, D. 1993b. An overview of the TP++ transport protocol project. In A. TANTAWY Ed., *High Performance Networks—Frontiers and Experience*, Chapter 8, pp. 157–176. Boston, MA: Kluwer Academic Publishers.

FELDMEIER, D. 1993c. A survey of high performance protocol implementation techniques. In A. TANTAWY Ed., *High Performance Networks—Technology and Protocols*, Chapter 2, pp. 29–50. Boston, MA: Kluwer Academic Publishers.

FELDMEIER, D. AND BIERSACK, E. 1990. Comparison of error control protocols for high bandwidth-delay product networks. In M. JOHNSON Ed., *Protocols for High-Speed Networks, II* (Palo Alto, CA, Nov. 1990), pp. 271–295. North-Holland Publ., Amsterdam, The Netherlands.

FLOYD, S. 1996. Issues of TCP with SACK. Technical report (Jan.), Information and Computing Sciences Division, Lawrence Berkeley Laboratory, Berkeley, CA. `ftp://ftp.ee.lbl.gov/papers/issues_sa.ps.Z`.

FLOYD, S. AND JACOBSON, V. 1993. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking 1*, 4 (Aug.), 397–413.

FLOYD, S., JACOBSON, V., LIU, C., MCCANNE, S., AND ZHANG, L. 1995. Reliable multicast framework for light-weight sessions and application level framing. In *ACM SIGCOMM '95* (Cambridge, MA, Sept. 1995). `ftp://ftp.ee.lbl.gov/papers/srm.sigcomm.ps.Z`.

FREEMAN, R. 1995. *Practical Data Communications*. Wiley.

GOLDEN, E. 1997. TRUMP: Timed-reliability unordered message protocol. MS Thesis, CIS Dept., University of Delaware.

HAAS, Z. 1990. A communication architecture for high-speed networking. In *IEEE INFOCOM* (San Francisco, CA, June 1990), pp. 433–441.

HAAS, Z. AND AGRAWAL, P. 1997. Mobile-TCP: an asymmetric transport protocol design for mobile systems. In *International Conference on Communications* (Montreal, Quebec, Canada, June 1997). IEEE.

HAN, R. AND MESSERSCHMITT, D. 1996. Asymptotically reliable transport of multimedia/graphics over wireless channels. In *Multimedia Computing and Networking 1996*, Volume 2667 (San Jose, CA, Jan. 1996), pp. 99–110.

HEIDEMANN, J. 1997. Performance interactions between P-HTTP and TCP implementations. *Computer Communication Review 27*, 2 (April).

HENDERSON, T. 1995. Design principles and performance analysis of SSCOP: A new ATM adaptation layer protocol. *Computer Communication Review 25*, 2 (April).

IREN, S. Network-conscious image compression. PhD Dissertation, CIS Dept., University of Delaware, (in progress).

IREN, S., AMER, P., AND CONRAD, P. 1998. Network-conscious compressed images over wireless networks. In *5th International Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services (IDMS'98), Oslo, Norway*, Volume 1483 of *Lecture Notes in Computer Science*, pp. pp 149–158. Springer Verlag.

ISO. 1989. International standard 7498-2 – information processing systems, open systems interconnection, basic reference model, part 2: Security architecture. .

ITU-T. 1994a. Recommendation Q.2110 – B-ISDN ATM adaptation layer - service specific connection oriented protocol (SSCOP). (July).

ITU-T. 1994b. Recommendation X.200 – information technology, open systems interconnection - basic reference model. (July).

ITU-T. 1995a. Recommendation I.365.3 – B-ISDN ATM adaptation layer sublayers: service specific coordination function to provide the connection-oriented transport service (SSCF-COTS). (Nov.).

ITU-T. 1995b. Recommendation Q.2144 – B-ISDN signalling ATM adaptation layer (SAAL) - layer management for the SAAL at the network node interface (NNI). (Oct.).

ITU-T. 1995c. Recommendation X.214 – information technology, open systems interconnection, transport service definition. (Nov.).

JACOBSON, V. 1988. Congestion avoidance and control. In *ACM SIGCOMM '88* (Stanford, CA, Aug. 1988). `ftp://ftp.ee.lbl.gov/papers/congavoid.ps.Z`.

JAIN, R. 1989. A delay-based approach for congestion avoidance in interconnected heterogeneous computer networks. *Computer Communication Review 19*, 5 (Oct.), 56–71. `www.cis.ohio-state.edu/~jain/papers/delay.ps`.

KARN, P. AND PARTRIDGE, C. 1987. Improving round-trip time estimates in reliable transport protocols. In *ACM SIGCOMM '87* (Stowe, VT, Aug. 1987).

KENT, C. AND MOGUL, J. 1987. Fragmentation considered harmful. *Computer Communication Review 17*, 5, 390–401.

KESHAV, S. Packet-pair flow control. To appear in IEEE/ACM Trans. on Networking.

KESHAV, S. 1991. A control-theoretic approach to flow control. In *ACM SIGCOMM '91* (Zurich, Switzerland, Sept. 1991), pp. 3–15.

KESHAV, S. AND MORGAN, S. 1997. SMART retransmission: Performance with overload and random losses. In *IEEE INFOCOM* (Kobe City, Japan, April 1997).

LA PORTA, T. AND SCHWARTZ, M. 1991. Architectures, features, and implementation of high-speed transport protocols. *IEEE Network Magazine*, 14–22.

LIN, S. AND COSTELLO, D. 1982. *Error Control Coding*. Prentice Hall.

Liskov, B., Shrira, L., and Wroclawski, J. 1990. Efficient at-most-once messages based on synchronized clocks. In *ACM SIGCOMM '90* (Philadelphia, PA, Sept. 1990), pp. 41–49.

Lundy, G. and Tipici, H. 1994. Specification and analysis of the SNR high-speed transport protocol. *IEEE Trans. on Networking 2*, 5 (October), 483–496.

Marasli, R. 1997. Partially ordered and partially reliable transport protocols: Performance analysis. PhD Dissertation, CIS Dept., University of Delaware.

Marasli, R., Amer, P., and Conrad, P. 1996. Retransmission-based partially reliable services: An analytic model. In *IEEE INFOCOM* (San Fransisco, CA, March 1996). www.cis.udel.edu/~amer/PEL/poc/postscript/infocom96.ps.

Marasli, R., Amer, P., and Conrad, P. 1997. An analytic model of partially ordered transport service. *Computer Networks and ISDN Systems 29*, 6 (May), 675–699.

Martin, J. and Leben, J. 1992. *DECnet Phase V: An OSI Implementation.* Digital Press.

Mathis, M., Mahdavi, J., Floyd, S., and Romanow, A. 1996. TCP selective acknowledgment options. RFC 2018 (Oct.).

McAuley, D. 1990. Protocol design for high speed networks. PhD Dissertation, University of Cambridge, Tech. Report No. 186.

McCanne, S., Jacobson, V., and Vetterli, M. 1996. Receiver-driven layered multicast. In *ACM SIGCOMM '96* (Palo Alto, CA, Aug. 1996).

McNamara, J. 1998. *Technical Aspects of Data Communication* (3rd ed.). Digital Press.

Nagle, J. 1984. Congestion control in IP/TCP internetworks. RFC 896 (Jan.).

Netravali, A., Roome, W., and Sabnani, K. 1990. Design and implementation of a high speed transport protocol. *IEEE Transactions on Communications 38*, 11, 2010–2024.

Partridge, C., Hughes, J., and Stone, J. 1995. Performance of checksums and crcs over real data. In *ACM SIGCOMM '95* (Cambridge, MA, Sept. 1995).

Parulkar, G. and Turner, J. 1989. Towards a framework for high speed communication in a heterogeneous networking environment. In *IEEE INFOCOM* (Ottawa, Canada, April 1989), pp. 655–667.

Piscitello, D. and Chapin, A. 1993. *Open Systems Networking, TCP/IP and OSI* (1st ed.). Addison-Wesley.

Porta, T. L. and Schwartz, M. 1993a. The multistream protocol: A highly flexible high-speed transport protocol. *IEEE Journal of Selected Areas in Communications 11*, 4 (May), 519–530.

Porta, T. L. and Schwartz, M. 1993b. Performance analysis of MSP: A feature-rich high-speed transport protocol. *IEEE Trans. on Networking 1*, 6 (Dec.), 483–496.

Postel, J. 1980. User datagram protocol. RFC 768 (Aug.).

Postel, J. 1981. Transmission control protocol. RFC 793 (Sept.).

Prue, W. and Postel, J. 1987. Something a host could do with source quench: The source quench introduced delay (SQuID). RFC 1016 (July).

Ramakrishnan, K. and Jain, R. 1988. A binary feedback scheme for congestion avoidance in computer networks with a connectionless network layer. In *ACM SIGCOMM '88* (Stanford, CA, Aug. 1988), pp. 303–313.

Robertson, D. 1996. *Accessing Transport Networks: MPTN and AnyNet Solutions.* McGraw-Hill.

Rose, M. and Cass, D. 1987. ISO transport service on top of the TCP, version:3. RFC 1006 (May).

Sanders, R. and Weaver, A. 1990. The Xpress transfer protocol (XTP) — a tutorial. *Computer Communication Review 20*, 5 (Oct.), 67–80.

Sanghi, D. and Agrawala, A. 1993. DTP: An efficient transport protocol. In S. Raghavan, G. Bochmann, and G. Pujolle Eds., *Computer Networks, Architecture and Applications*, pp. 171–180. North Holland, Amsterdam.

Schulzrinne, H. 1996. RTP profile for audio and video conferences with minimal control. RFC 1890 (Jan.).

SCHULZRINNE, H., CASNER, S., FREDERICK, R., AND JACOBSON, V. 1996. RTP: a transport protocol for real-time applications. RFC 1889 (Jan.).

SHENKER, S., PARTRIDGE, C., AND GUERIN, R. 1997. Specification of guaranteed quality of service. RFC 2212 (Sept.).

SHENKER, S. AND WROCLAWSKI, J. 1997. General characterization parameters for integrated service network elements. RFC 2215 (Sept.).

SMITH, W. AND KOIFMAN, A. 1996. A distributed interactive simulation intranet using RAMP, a reliable adaptive multicast protocol. In *Fourteenth Workshop on Standards for the Interoperability of Distributed Simulations* (Orlando, FL, March 1996). www.tasc.com/simweb/papers/disramp/index.html.

STALLINGS, W. 1997. *Data and Computer Communications* (5th ed.). Prentice Hall.

STALLINGS, W. 1998. *High-Speed Networks: TCP/IP and ATM Design Principles.* Prentice Hall.

STEVENS, W. 1994. *TCP/IP Illustrated: The Protocols*, Volume 1. Addison-Wesley, Reading, MA.

STEVENS, W. 1996. *TCP/IP Illustrated*, Volume 3. Addison-Wesley, Reading, MA.

STEVENS, W. 1997. TCP slow start, congestion avoidance, fast retransmit, and fast recovery algorithms. RFC 2001 (Jan.).

STEVENS, W. 1998. *UNIX Network Programming: Networking APIs, Sockets, and XTI* (2nd ed.). Prentice-Hall.

STRAYER, T., DEMPSEY, B., AND WEAVER, A. 1992. *XTP – The Xpress Transfer Protocol.* Addison-Wesley Publishing Company.

STRAYER, T. AND WEAVER, A. 1988. Evaluation of transport protocols for real-time communications. Technical Report TR-88-18 (June), CS Depart. Univ. of Virginia, Charlottesville, VA. ftp://uvacs.cs.virginia.edu/pub/techreports/CS-.ps.Z.

TANENBAUM, A. 1996. *Computer Networks* (3rd ed.). Prentice-Hall.

THAI, K., CHASSOT, C., FDIDA, S., AND DIAZ, M. 1994. Transport layer for cooperative multimedia applications. Technical Report 94196 (May), Centre National de la Recherche Scientifique (CNRS), France.

WALRAND, J. 1991. *Communication Networks: A First Course.* Aksen Associates.

WATSON, R. 1989. The Delta-T transport protocol: Features and experience. In *14th Conference on Local Computer Networks* (Minneapolis, Minnesota, Oct. 1989). IEEE.

WEAVER, A. 1994. The Xpress transfer protocol. *Computer Communications 17*, 1 (Jan.), 46–52.

WILLIAMSON, C. AND CHERITON, D. 1989. An overview of the VMTP transport protocol. In *14th Conference on Local Computer Networks* (Oct. 1989). IEEE.

WROCLAWSKI, J. 1997. Specification of the controlled-load network element service. RFC 2211 (Sept.).

YANG, C. AND REDDY, A. 1995. A taxonomy for congestion control algorithms in packet switching networks. *IEEE Network*, 42–48.

YAVATKAR, R. AND BHAGWAT, N. 1994. Improving end-to-end performance of TCP over mobile internetworks. In *Mobile'94 Workshop on Mobile Computing Systems and Applications* (Dec. 1994).

ZHANG, L. 1986. Why TCP timers don't work well. In *ACM SIGCOMM '86* (Stowe, Vermont, Aug. 1986), pp. 397–405.