XEROX

Xerox System
Integration Standard

Internet Transport Protocols

XEROX

Xerox System Integration
Standard

Internet Transport Protocols

XSIS 028112
December 1981

Xerox Corporation
Stamford, Connecticut 06904

Notice

This *Xerox System Integration Standard* describes the Internet Transport Protocols—the family of internetwork packet transport protocols used uniformly across the variety of communication media, digital processors, and office applications in Xerox Network Systems.

1. This standard includes subject matter relating to patent(s) of Xerox Corporation. No license under such patent(s) is granted by implication, estoppel, or otherwise, as a result of publication of this specification.

2. This standard is furnished for informational purposes only. Xerox does not warrant or represent that this standard or any products made in conformance with it will work in the intended manner or be compatible with other products in a network system. Xerox does not assume any responsibility or liability for any errors or inaccuracies that this document may contain, nor have any liabilities or obligations for any damages, including but not limited to special, indirect, or consequential damages, arising out of or in connection with the use of this document in any way.

3. No representations or warranties are made that this specification, or anything made in accordance with it, is or will be free of any proprietary rights of third parties.

Preface

This document is one of a family of publications that describes the network protocols underlying Xerox Network Systems.

Xerox Network Systems comprise a variety of digital processors interconnected by means of a variety of transmission media. System elements communicate both to transmit information between users and to economically share resources. For system elements to communicate with one another, certain standard protocols must be observed.

This document is not a rigorous specification in the formal sense; one cannot prove the correctness of the protocols from this standard, nor perform validation. Rather, the standard describes the protocols and provides the rationale behind many of their features; it provides the necessary information for designing an implementation.

Comments and suggestions on this document and its use are encouraged. Please address communications to:

Xerox Corporation
Office Products Division
Network Systems Administration Office
3333 Coyote Hill Road
Palo Alto, California 94304

Table of contents

Figures

1

Introduction

1.1 Purpose

The Xerox Network Systems (NS) Internet Transport Protocols provide the architectural foundation for Xerox' distributed system. The protocols are an open-ended set of *internetwork packet transport protocols* used uniformly across a variety of communication media, digital processors, and office applications, all of which may vary from installation to installation and from time to time. A communication system built using these protocols permits system elements to exchange data and its associated semantics. Higher-level protocols built on top of the Internet Transport Protocols provide mechanisms for describing data, exchanging files, sending documents to printers, etc.

This document specifies the Internet Transport Protocols in sufficient detail to allow systems to be built that can communicate with Xerox' Network Systems.

1.2 Fundamental concepts

The Internet Transport Protocols embody the fundamental principles associated with store-and-forward internetwork packet communication. The protocols provide for the transport of data across an interconnection of *networks* (*communication media* or *transmission systems*, as they are also called); this interconnection is called an *internetwork* or *internet*.

The fundamental unit of information flow in the system is the media-, processor-, and application-independent *internet packet*. An internet packet contains some control information plus some data, where the data may range from a few bits to several thousand bits. (The format of the internet packet is described in detail in Section 3.) Internet packets are *routed* through the internetwork as *datagrams* via store-and-forward system elements called *internetwork routers*. Internetwork routers connect transmission systems together. Internet packets are routed from source address to destination address, potentially through a variety of transmission systems, each *encapsulating* and *decapsulating* the packet in its own medium-specific way. Internet packets are datagrams, that is, the internet routing machinery treats each one independently. The internet gives its *best effort* to deliver an internet packet, but it cannot guarantee that they are delivered once and only once, nor that they will be delivered in the same order they were transmitted.

It is impractical to define a universal transport protocol sufficient for the special needs of all possible clients. We do not attempt such a task; instead we specify a number of protocols based on the internet packet. These protocols accomplish certain general communication functions, such as delivery of sequenced data and exchange of routing information. The protocol architecture permits implementors with special needs to escape into their own protocols. It is expected that as these special needs become better understood general protocols to deal with them will be specified.

It is a fortunate property of communication systems that the functions usually needed are naturally structured in such a way that one builds on the next in a layered way. We have defined a number of protocol levels, based on how information is layered within a packet. This taxonomy is based on packet formats and, in general, maps onto a taxonomy based on function. Since the protocol architecture is very general and open-ended, it is possible to map different functions into levels in ways other than that described here.

The fundamental principle of layered protocol design is that each level defines a *type* field that is interpreted by the next higher layer, providing a bridge between the two levels. This principle permits a protocol architecture that is open-ended and which defines a layered tree of protocols. The levels are defined from the perspective of the internet packet, and are the following:

At the bottom level, *level zero*, there is a need to physically transmit data from one point to another. This level is highly dependent on the particular transmission medium involved and is touched on only briefly in this document. There may be many different level zero protocols, and some of them may contain many internal levels.

At *level one* there is a need to decide where the data should go. This level is concerned with how to address a source and a destination, and how to choose the correct transmission medium to use in order to route the packet toward its goal. There is only one level one protocol, and it defines the internet packet and rules for its delivery as a datagram.

At *level two* there is a need to give structure to a stream of related packets. This involves retransmission, sequencing, duplicate suppression, and flow control. It is clearly recognized that some applications need far less structure of this sort than others, and the protocol architecture allows for alternate implementations of the level two protocols. This document describes several level two protocols. Level one and two protocols are typically called *transport protocols.*

At *level three* the protocols have less to do with communication and more with the content of data and the control or manipulation of resources. Such protocols are often called *control protocols.* Here one finds protocols for talking about remotely accessible procedures, files, printing functions, terminals, or displays. This document describes no level three protocols.

Protocols above level three have little to do with communication, but instead deal with data structures and formats. They are often called *application protocols.*

Figure 1.1 illustrates this layered tree of protocols (packet formats). It shows how the protocols are structured, but does not show how some user-initiated function is performed using these protocols.

The scope of this document is level one and two protocols.

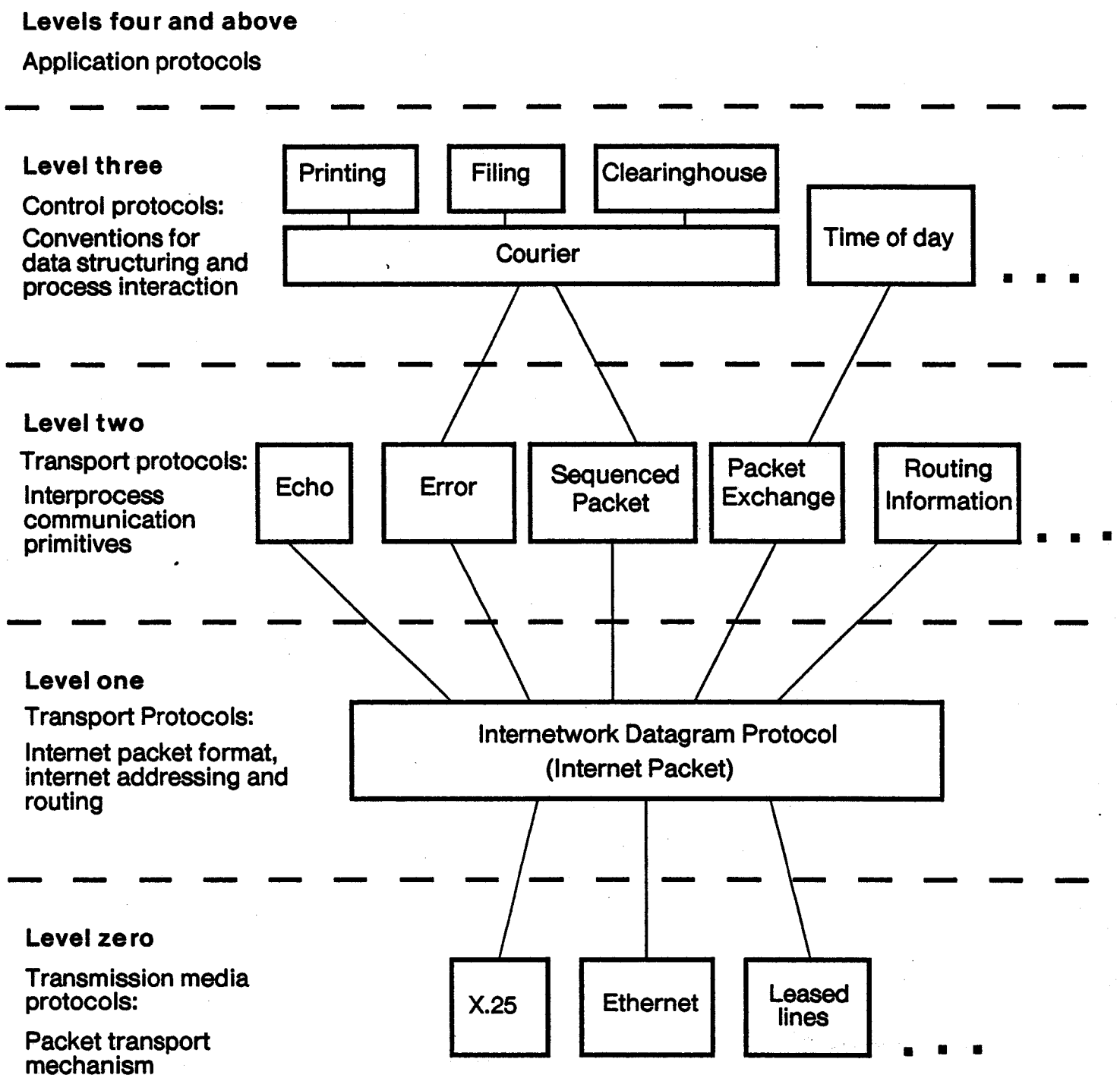


Figure 1.1 The layered tree of protocols

We assume that both the incorrect delivery of data and the delivery of incorrect data are very serious matters. Therefore, steps are taken to greatly reduce the probability of this happening. It is further assumed that internet packets may be delayed in their delivery for periods averaging a few hundreds of milliseconds, and that an exceptional packet may be delayed about a minute. The transmission systems presently under consideration are expected to operate at bandwidths from the low kilobit range to the megabit range. It is assumed that packets carried through such systems are subject to:

- size limitations (~1,024 to 1,048,560 bits)
- propagation, queuing, and routing delays (~ 1 ms to 60 sec)
- duplication (due to retransmission, rerouting, and lost acknowledgements)
- reordering (due to retransmission and adaptive routing)
- undetected damage (~ 1 error in 10^3 to 10^{12} bits)
- unannounced loss (~ 1 packet per 10 to 1,000,000)
- misrouting (due to internal malfunction)
- unauthorized perusal (surveillance)
- counterfeiting (sabotage)

The Internet Transport Protocols specify how various communication functions are accomplished in internets made of such transmission systems. The transport functions provided by the level one and two protocols include: addressing and routing, congestion control, flow control, error, detection, retransmission, duplicate suppression, sequencing and packetizing. Security against surveillance and sabotage is a higher-level function.

1.3 Terminology

In discussing packet protocols, the following terminology is generally useful. A packet has a *source* and a *destination.* A flow of data has a *sender* and a *receiver,* recognizing that to support a flow of data, some packets (typically acknowledgements) will be sourced at the receiver and destined for the sender. A *connection* is a transient association between two communicators, during which time data is reliably transferred between them. A connection generally has a *listener* and an *initiator.* A service has a *supplier* and a *consumer.* A system element with services is called a *server.* It is very useful to treat these as orthogonal descriptors of the participants in a communication. Of course, a server is usually a listener and the source of data-bearing packets is usually the sender. Data may flow in both directions over a connection, and in some rare cases a connection may even have two initiators.

A *network* is a transmission medium configured to carry internet packets. A *transmission medium* is any communication equipment configured to carry data. A *host* is any system element supplied with a unique 48-bit host identifier, connected to a network, and supporting the communication protocols. A *socket* is uniquely identified within a host, and is a source and/or destination of packets. A socket is simply an object within a host, to which packets can be delivered, and from which packets may be transmitted. Like a telephone, which is able both to originate and receive calls, a socket is inherently a bidirectional structure, able both to send and receive packets.

A network can be a broadcast network, a multicast network or a point-to-point network. A *broadcast network* is one in which the routing algorithms of the network make it possible to transmit a packet to all hosts connected to the network. A *multicast network* is one in which the routing algorithms of the network make it possible to transmit a packet to some subset of the hosts connected to the network. *Point-to-point networks* only permit the routing of packets from one host to another. The Ethernet is an example of a broadcast and multicast network, while a phone line and Telenet are examples of a point-to-point network. An *internetwork*, or simply *internet*, is an interconnection of networks that carry internet packets.

The Internet Transport Protocols support communication between sockets within the internet, in much the same way that the post office transfers letters between post office boxes. The sockets may be on the same host, or on hosts on the same network, or on hosts on different networks. The logical function of switching (routing) internet packets between sockets (that are on the same host), between sockets and networks (the host is the source or destination of the packet), and between networks themselves (in an internetwork router) is abstractly captured in a *router*. A router is usually implemented in software and resides in *every* system element; it may provide only a subset of the full switching capabilities. For example, a router in a workstation or file server will typically route packets between local sockets, between sockets and networks (for transmission), and between networks and sockets (for reception). A router in a store-and-forward switching node, in addition to doing the above, spends most of its time routing packets between networks—hence the name *internetwork router*.

Every router maintains a *routing table* that contains the delay to some number of networks that are one or more hops away, and the identity of an internetwork router by which they can be reached. The routing table may be updated dynamically by using the Routing Information Protocol (see Section 4).

Most internets cover areas that are geographically dispersed. To facilitate discussion of the internet, its topology is given some *informal* structure. A *campus* is a building or collection of adjacent buildings occupied by a number of system elements serving a customer. Communication among system elements is of two kinds: *intracampus* and *intercampus*. Intracampus communication is typically accomplished through (the interconnection of) high-bandwidth packet communication systems—usually Ethernets. Intercampus communication uses common carrier data transmission facilities subject to FCC regulation and tariffs. Intercampus communication is conducted though a variety of common carrier facilities. The choice of facility depends on the volume, frequency, and dispersion of communications. As lower-cost, higher-rate, modemless digital transmission facilities become available (generally at 50 kilobits per second and up), they will be employed to carry higher volumes of data. As public packet-switching facilities become available, they will be employed to carry higher dispersions of low-volume office communication.

There are generally two kinds of incompatibility in communication: *media incompatibility* and *protocol incompatibility*. The first arises from communication using the Internet Transport Protocols *through* different communication media (as discussed above). The second kind arises from communication *with* non-Network Systems components—often referred to as *foreign devices*—for example, with a Xerox 850.

Gateway services reside in various system elements, and capture the incompatibilites between Network Systems and non-Network Systems. Foreign devices do not communicate according to protocols consistent with Xerox' Network Systems architecture, but rather according to their

own protocols. Gateways, therefore, communicate with foreign devices using *their* protocol and perform protocol translation or conversion at *any* of the necessary levels. Gateways are not a subject of this document.

Figure 1.2 illustrates a typical internet, and shows how an internet packet is routed through the internet using a store-and-forward algorithm.

1.4 Design and implementation considerations

The Internet Transport Protocols software within a host provide client processes interfaces to level one and some level two protocols. Application programs running as processes under an operating system and support software may use the Internet Transport Protocols facilities via different grades of service, depending on their requirements for generality and efficiency. These grades of service are offered in the following levels:

- datagram packet (routing internet packets from socket to socket)
- packet exchange (request/reply packet exchange)
- packet stream (sequencing, retransmission, flow control)
- byte stream (packetizing)

Figure 1.3 illustrates how the layered protocol architecture might be implemented within an operating system environment in a host. The structure will be better understood once the reader has become more familiar with the protocols. In general, there is no one-to-one relation between sockets and host processes, and it is considered appropriate for a process to use several sockets, or for several processes to share a single socket; the operating system generally influences this choice. The implementation of a protocol will often be referred to as a *protocol package*, and software that makes use of its facilities is called its *client*.

At level zero there are a number of *network drivers* (constructed from transmission medium drivers) that know their network number and transport internet (and perhaps other) packets over the transmission medium. At level one is the Internet Transport Protocols router that routes packets between sockets, sockets and networks, and networks themselves. Some sockets will be well-known, while others will be temporary. (It is possible to implement other protocol architectures in parallel, by having "routers" for them and a protocol *dispatcher* between the "routers" and "networks".) At level two, a number of protocols are implemented in a cooperative fashion to provide a meaningful service. For example, at well-known socket 1, the Routing Information and Error Protocols would be implemented; at some socket, *s*, the Sequenced Packet Protocol in conjunction with the Error Protocol provides a byte stream capability; and so on. At level three, other protocols which make use of the Internet Transport Protocols services are implemented. This structure can be considerably simplified if there is only one network driver and/or only a few sockets. The ease with which an operating system supports concurrent processes and interprocess communication dramatically influences the overall design and performance of the system.

The important observation is that all internet packets are delivered to some socket, and that it is the client of the socket that interprets one or more level two formats.

Internetwork routers and gateways need not occupy entire system elements, but may reside in the same system element providing access to other shared resources.

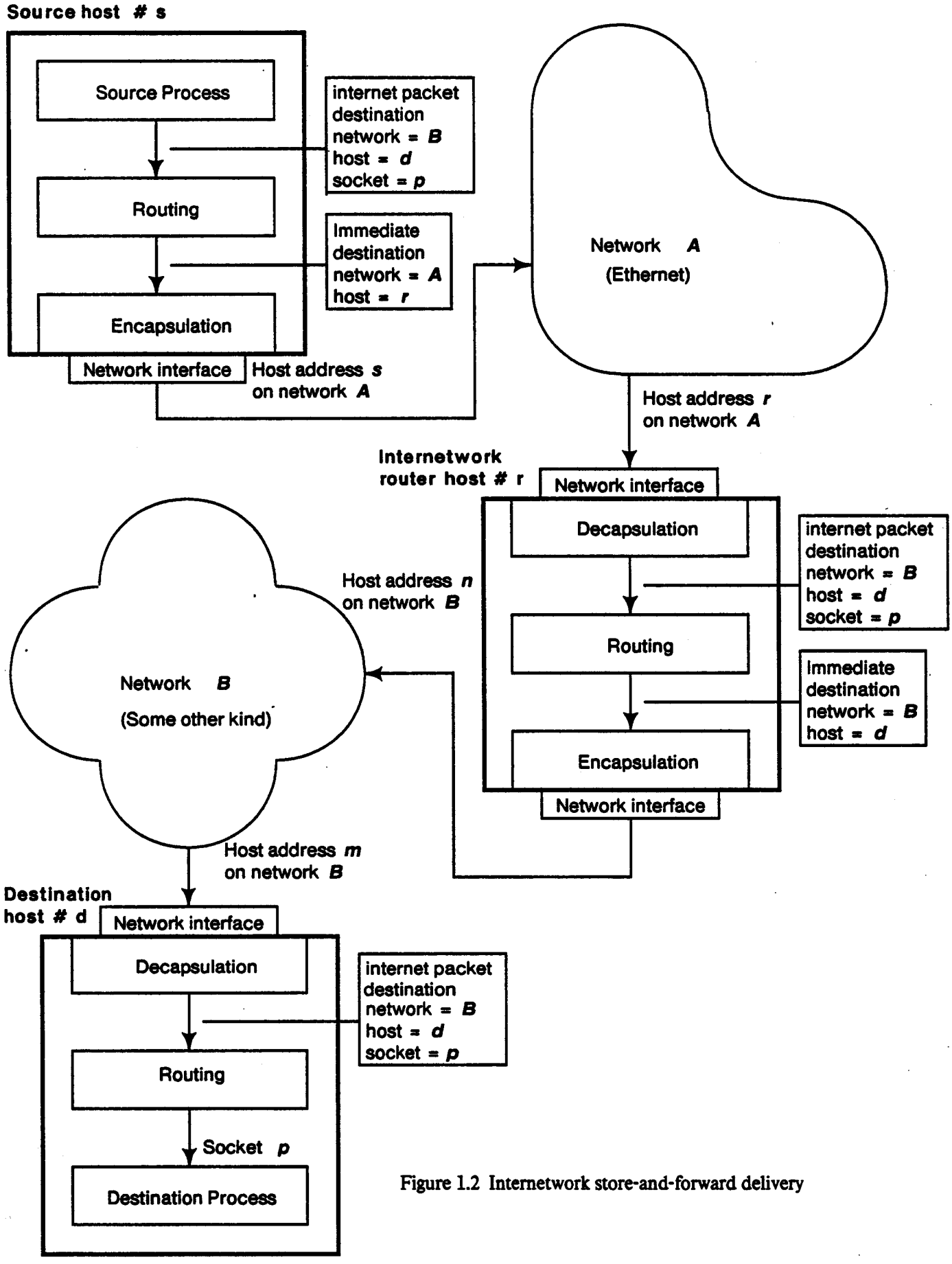
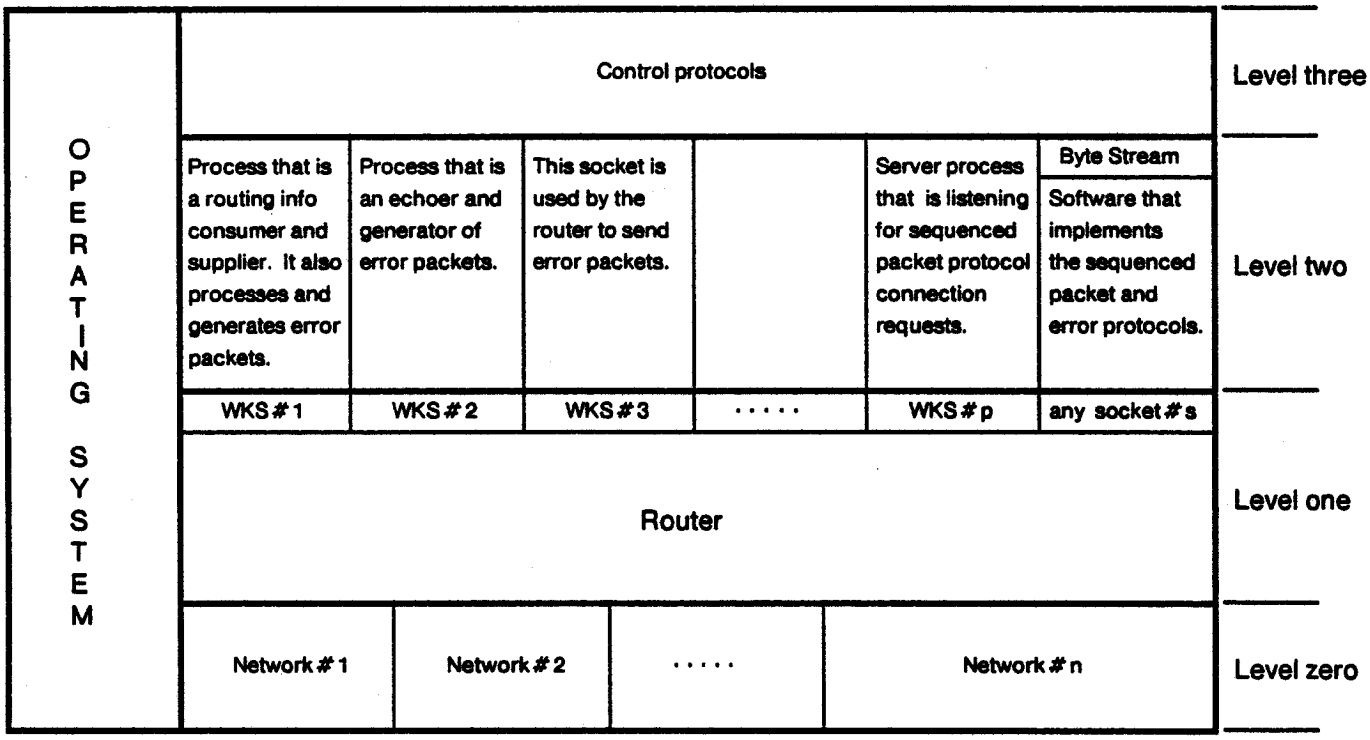


Figure 1.2 Internetwork store-and-forward delivery



WKS = Well-Known Socket

Figure 1.3 An internet transport protocols implementation

1.5 History and context

The Internet Transport Protocols embody principles that evolved from many years of operational experience with the Pup internetwork, a research prototype developed at the Xerox Palo Alto Research Center [2]. The NS Internet Transport Protocols resemble the Internet Protocol (IP) and Transmission Control Protocol (TCP) developed by DARPA for the Department of Defense [11; 12; 4].

1.6 Relationship to ISO Open Systems Interconnection Reference Model

The protocol architecture underlying Xerox' Network Systems is layered, and meets the basic objectives of the ISO Open Systems Interconnect Reference Model [17]. Where the reference model allows selection of certain options or interpretations regarding functions or relationships, this specification describes those selections made by Xerox. Level zero corresponds to layers 1, 2 and 3a, the physical, data link and network layers; level one to layer 3b, the internet layer; and level two to layer 4, the transport layer. Courier corresponds to layer 6, the presentation layer. Higher-level protocols like filing, printing and Clearinghouse correspond to layer 7, the

application layer. The Xerox Network System has no protocol corresponding to layer 5, the session layer.

1.7 Document structure

Section 2 of this document describes the requirements for level zero transmission media protocols without explicitly describing any specific ones, and Sections 3 through 8 define the Internet Datagram Protocol, the Routing Information Protocol, the Error Protocol, the Echo Protocol, the Sequenced Packet Protocol, and the Packet Exchange Protocol. The description of each protocol includes a sub-section containing implementation notes that describe salient features and caveats. Appendix A lists references, Appendices B through E define various "number" assignment procedures, and Appendix G defines a connection termination protocol.

2

Level zero: Transmission media protocols

2.1 Introduction

The function of a *transmission medium protocol* is to move data across a transmission medium. Examples of a transmission medium are the Ethernet, DDS, and a 9600 bps (bits per second) line using HDLC protocols. This level is assumed to include all of the transmission medium-specific protocols. The network driver associated with each medium transports level one packets from one router to another. X.25 [6] is also treated as a transmission medium protocol by the Internet Transport Protocols. In fact, any protocol used to transport level one (internet) packets across a communication system is viewed as a level zero protocol [15].

The level zero transmission medium protocol is documented with the specification of a particular transmission medium. In the case of the Ethernet, the description may be found in the document *The Ethernet, A Local Area Network: Data Link Layer and Physical Layer Specifications* [8]. (Note that the Ethernet Type for the NS Internet Transport Protocols' internet packet is 3000 octal. See Figure 2.1 and Appendix B.)

At level zero, the level one packet is treated as data. The level one packet is interpreted only by the next higher level (see Section 3). Level one packets are to be encapsulated for transmission in each packet, block, or frame of level zero format as per the conventions of the transmission medium protocol. Upon reception, the transmission medium protocol specifies the inverse process of decapsulation, which preserves the structure of the level one packet.

All level one (and higher) packet formats in this document are drawn as they appear in the memory of a processor that has a 16-bit word, and where the first word of the packet is in memory location *m*, and the next word in location *m+1*. The bits (or bytes) within a word are numbered left to right. (Figure 3.1 shows an internet packet.) The process of encapsulation and decapsulation *must* preserve this logical structure.

A well-designed transmission medium protocol should have a mechanism for identifying the type of data it encapsulates—for example, the Internet Transport Protocols' internet packet, a Pup, or a DoD Internet packet—for easy demultiplexing within the receiving host. This permits the same transmission medium to be simultaneously used to support many different types of level one formats and protocols. This feature is not found in all commercially available transmission systems; in those cases, the level zero protocol must enhance the basic

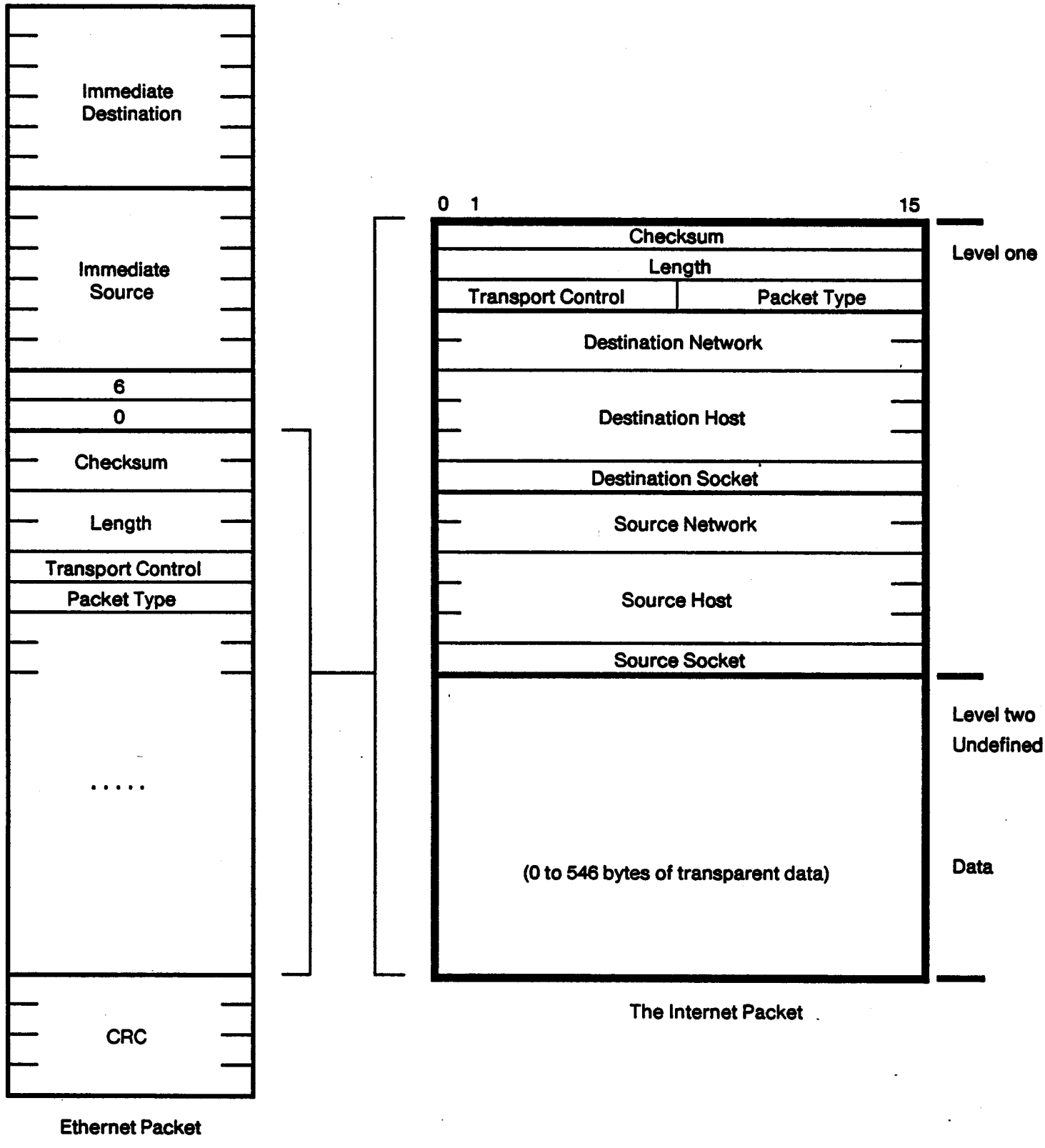


Figure 2.1 An internet packet encapsulated in an Ethernet packet

functions to provide this feature. In this section we will use the Internet Transport Protocols' internet packet as an example of a level one packet.

Packet *encapsulation* consists of transforming an internet packet in whatever fashion is necessary to permit the entire internet packet to pass through the transmission medium transparently as "data". This generally involves simply adding transmission medium-dependent headers and/or trailers for addressing, error detection or correction, etc., but could also include transmission medium-specific fragmentation and retransmission, and encoding of data in various ways such as bit compression (for transmission over phone lines, for example). The process of encapsulation is specific to a pair of level one and level zero protocols. In the Internet Transport Protocols, the transmission medium is expected to transport the internet packet only with high probability.

Prior to encapsulation, the level one implementation determines an *immediate destination host* from the internet packet *destination host* (see Section 3). This will be either the final destination host (if that host is directly connected to the transmission medium over which the internet packet is about to be transported), or an internetwork router through which it is believed (by some sort of routing function) the final destination can be reached.

In order to physically transmit an internet packet to an immediate destination host, the host must be addressed according to the addressing conventions of that transmission medium. It is likely that transmission media other than the Ethernet will have addressing schemes different from that employed by the Internet Transport Protocols. In such cases, a transmission medium-specific address must be derived from the 48-bit internet packet destination host number [7]. In addition to point-to-point addressing, the Internet Transport Protocols support broadcast and multidestination delivery (see Section 3). Transmission media not directly supporting these forms of addressing may or may not be augmentable via functions provided at level zero to transport such internet packets.

Figure 2.1 shows how an internet packet (see Section 3 for details) is encapsulated in an Ethernet packet. The internet packet is converted into a sequence of 8-bit bytes by going left to right within a word, and going from word m to word $m+1$ within the packet. Since an Ethernet packet must have a minimum of 46 data bytes [8], if the level one packet is smaller than this, then the process of encapsulation must add padding at the end of the level one packet to account for the difference. Since the internet packet has a length field of its own, the padding will be removed at the destination host.

When an internet packet is received over a transmission medium, it is first decapsulated by applying the inverse of the encapsulation transformation. In some cases, this may involve packet fragment reassembly, code conversion, padding removal, etc. The router receives the internet packet and delivers it to a local socket or routes it to another network. If the internet packet is routed to another network, it is reencapsulated and subsequently transmitted according to the conventions of this new transmission medium.

2.2 Implementation notes

The design of the Internet Transport Protocols implies that the internet treats each internet packet independently—that is, the internet is datagram in nature. However, not all constituent transmission media will be datagram in nature. In such cases, a circuit or virtual circuit will be set up for the transport of internet packets over the transmission medium. The heuristics that

determine when a (virtual) circuit must be set up and taken down are transmission medium-specific, since they must be based on the speed with which (virtual) circuits can be set up and taken down, and the cost of maintaining them.

The Internet Transport Protocols also specify a maximum packet lifetime for an internet packet within the internet (see Section 3). If an internet packet has remained on the "transmit queue" of a level zero network driver for longer than 1/10 this lifetime, it should be discarded. A network driver (the implementation of a level 0 protocol for a transmission medium) may, therefore, discard packets; this is consistent with the expectation that the internet gives its best effort to deliver a packet.

Network drivers must also ensure that those level one packets submitted for transmission that are addressed to the transmitting host itself (explicitly or implicitly) be delivered back to the host (explicitly by creating a copy, or implicitly by transmitting the packet on the transmission medium and then receiving it).

3

Level one: Internet datagram protocol

3.1 Introduction

The Internet Transport Protocols define only one level one protocol—the *Internet Datagram Protocol.* The function of the Internet Datagram Protocol is to address, route and deliver standard internet packets, each of which is treated as an independent entity with no relation to other internet packets traversing the system. The internet gives its best effort to deliver internet packets, but it cannot guarantee that they are delivered once and only once, nor that they will be delivered in the same order they were transmitted.

The format of the internet packet is shown in Figure 3.1. The internet packet is seen to consist of *Header* and *Data.* This is the format defined by the Internet Datagram Protocol. The header consists of three parts. Part one contains four fields related to controlling transmission of the internet packet. Parts two and three specify the address of the destination and source of the internet packet, respectively. These addresses are elaborated upon below; for now, it is enough that they uniquely define the communicating sockets in the entire internet.

3.2 Network addresses

The addressing conventions of the Internet Transport Protocols are examined first in order to illustrate the generality of the routing and delivery mechanism supported by this internetwork architecture. The internet packet has *Source Network Address* and *Destination Network Address* fields. The Internet Datagram Protocol says nothing about which process initiated the conversation, nor about the general direction of flow of data; it deals only with delivery of the packet at hand.

A *network address* is composed of three fields. The 32-bit *network number* uniquely identifies a network in an internet. The 48-bit *host number* identifies one of over four billion hosts, and is unique across *all* Network Systems processors ever manufactured. The 16-bit *socket number* uniquely identifies a socket within the operating system of a host. As we shall see, resources in hosts connected to multiple networks will have multiple network addresses, each one differing in the network number. The properties of the three fields of a network address are described below.

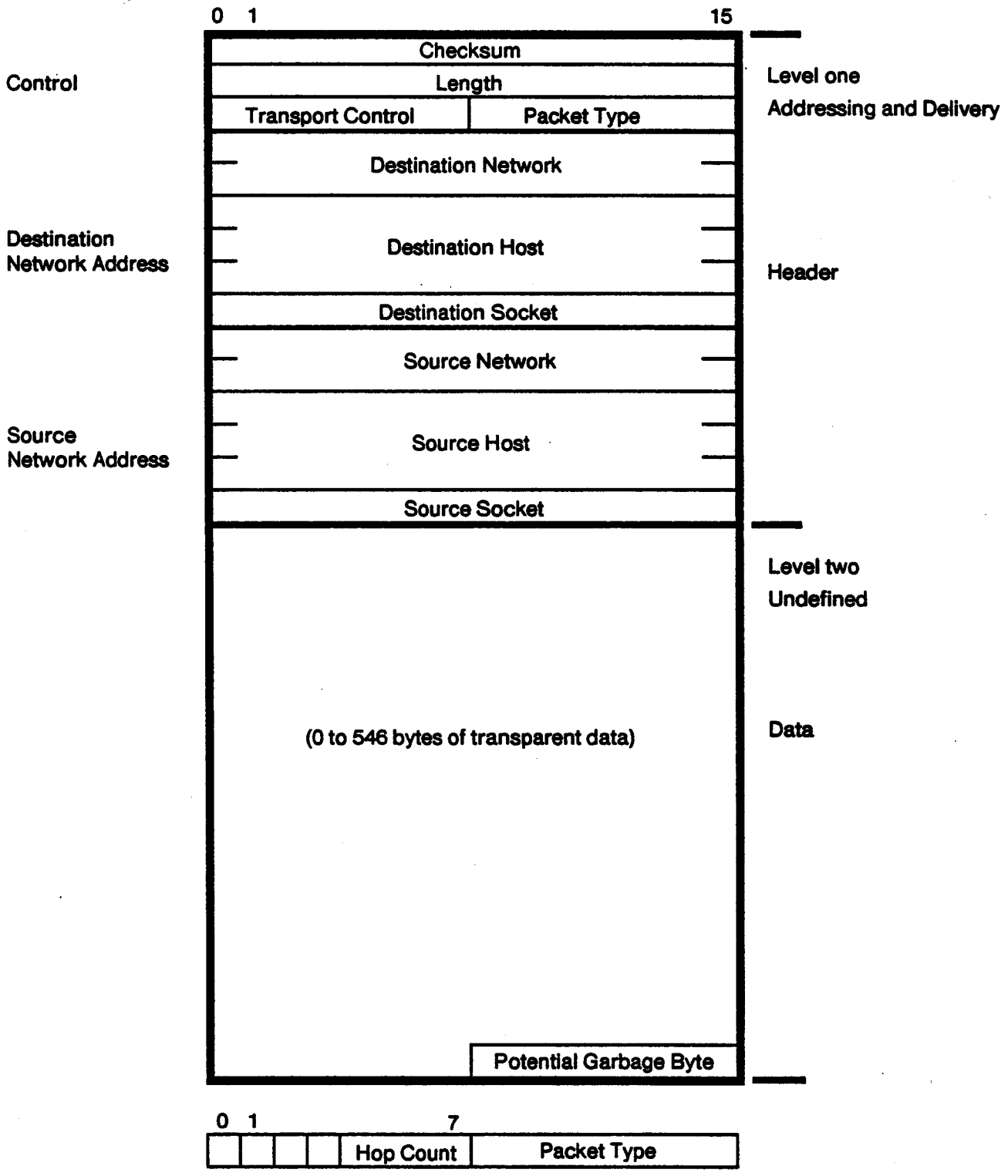


Figure 3.1 The internet packet

Host Numbers

Host numbers are *absolute*, and every system element must be assigned a unique 48-bit number independent of the network(s) to which it is connected. Ethernet addresses are identical to the Internet Datagram Protocols' 48-bit host numbers [8]. When a host is connected to more than one Ethernet, its 48-bit level zero Ethernet address on all Ethernets is equal to its 48-bit host number. The procedures for acquiring such numbers are specified in Appendix B.

Absolute host numbers have many advantages. In building large distributed systems it is helpful to be able to refer to a host's number independently of the network(s) to which it is currently (if at all) connected. Operating systems and application software can use this number in the generation of unique identifiers, for example [13]. Also, when a host is moved from one network to another, its host number does not change and no alterations need be made to the hardware—a substantial reduction in field service overhead. Of course, the network addresses of resources might change, thereby requiring rebinding of resource directories, etc., but this is a higher-level operation that must often be performed, and which can be automated through software procedures.

It is likely that the host number will be hardwired to some part of the machine—the backplane or one of the processor boards. If this piece of hardware is replaced, it is likely that the host number of the machine and its associated software will change. Reinitialization of the software may then be necessary, but this is something that would typically have to be done if any other characteristic of the hardware changed. It is important to note that a machine's host number may change, but no two machines may have the same number at the same time.

Xerox internets will, for the most part, consist of Ethernets, and it is for this reason that Ethernet addresses are identical to 48-bit host numbers. This is an optimization and a convenience, and in no way compromises the generality of the architecture [7]. In fact, the management of Ethernet addresses is considerably simplified. When an internet packet is encapsulated for transmission over an Ethernet, there is no table-driven overhead in performing the necessary address translation. As a consequence, procedures and mechanisms for creating and managing the translation table in every machine connected only to an Ethernet are eliminated. When internet packets traverse other communication networks, table-driven translation may be necessary.

Network Numbers

Since a host number uniquely identifies a particular host, the network number field is redundant but nevertheless serves the vital function of assisting internetwork routing. With the inclusion of the network number in the network address, it is sufficient for each router to know the path to each network, rather than to each host—a significant reduction in the amount of information that must be retained. A host may be connected to more than one network, but still has a unique identity, even though a socket in it has multiple network addresses. Therefore, network addresses are unambiguous but not unique. Sources or destinations of internet packets can, therefore, have more than one network address, but no two sources or destinations can have the same network address. An internet packet addressed to a host will contain the identity of the network to which the source believes the host is connected. Routers will attempt to route the internet packet to the host via this network. A shorter route via another network to which the host is connected will not be used. If no route to the specified network exists, then the packet will not be delivered, and the client must use another network

address. A higher-level binding mechanism must supply all the network addresses for a resource [10].

All networks within an internet must have unique network numbers if routing is to work at all. In addition, all networks should have unique network numbers across all internets to avoid the unnecessary operational problems of renumbering certain networks if internets are merged into a larger internet. The properties of a network are described in Section 4. The procedures for acquiring network numbers are specified in Appendix C.

Socket Numbers

A socket is inherently a bidirectional structure, able both to send and receive internet packets at the same address. All of the protocols described in this document use both the *send* and the *receive* side of sockets. Certain socket numbers are considered *well-known*, that is, the service performed by software using them is statically defined. Each system element supplying a specific well-known service does so at the same well-known socket. The assignment of values to existing well-known sockets is summarized in Appendix D, along with the procedures for acquiring new well-known socket numbers. All other socket numbers are ephemeral, that is, they may be reused.

3.3 Broadcast and multicast

The above discussion has implied that a host number field identifies precisely one host. The host numbering scheme in the Internet Datagram Protocol is actually more general than that. While it is true that each host has exactly one *physical* host identifier, a host number can also identify a *logical* host. A logical host is defined to be a group of hosts.

Multicast is the delivery of a packet to more than one destination. Multicast can be performed at either the internetwork or intranetwork level if the transmission medium supports the concept. The Internet Datagram Protocol supports internetwork multicast. *Broadcast* is a special case of multicast in which the multicast group consists of *all* the system elements in the internet. The need for a generalized multicast capability arises from the *anticipated* need for a more general addressing capability by applications such as multiple viewers of databases and packet-voice communication. Broadcast has been used in many situations to search for an object, or to inform all hosts of an event [3]. While we believe that all applications can be designed *without* a generalized multicast capability, its availabilty provides some additional performance improvements.

A *multicast group* is the list of intended recipients of the packet. There are many ways of specifying the multicast group: either by explicitly enumerating the destinations, or by an identifier that has a suitable encoding. In order to accept a multicast packet, every system element has an *acceptance filter* by which it determines whether the packet is intended for it or not. In the Internet Datagram Protocol, multicast groups are identified by logical host identifiers. One bit in the host number indicates whether the host number is a physical host number or a multicast identifier. The logical host number *all hosts* consisting of 48 ones is reserved for broadcast [8].

Physical host identifiers are assigned to system elements at manufacture time or by carefully controlled administrative procedures. Multicast identifiers are assigned by some dynamic resource allocation mechanism, or by administrative procedure, or both, depending on whether multicast identifiers are reusable or not.

At the Internet Datagram Protocol level, multicast involves communication between one source and many destinations. The participants—who may be geographically quite dispersed—must cooperate by using the same network address: a *multicast network address.* The socket number in a multicast network address could be statically defined for all multicast network addresses, but this would require receiving all packets for all multicast groups at one socket and then demultiplexing them using a level two multicast protocol. The Internet Datagram Protocol does not require that there be one socket number for all multicast network addresses. In fact, a multicast group could have a number of multicast network addresses, each one having a different socket number so that the demultiplexing of semantically different packets can be achieved easily. The restrictions on the value of the network number in a multicast network address will be discussed a little later.

Since all recipients of a multicast must receive the packet on the same socket, a set of well-known sockets must be reserved for multicast. Multicast well-known socket number assignment is subject to the same form of allocation mechanism as multicast identifiers. It is quite reasonable to have a somewhat centralized mechanism for the assignment of multicast identifiers and associated well-known socket numbers. Whether they are assigned by administrative procedure or via a dynamic resource allocation strategy is a matter of style, and a function of the rate at which new multicast network addresses are created, the number of multicast network addresses available, and the need to reuse them.

Since multicast refers to the delivery of a packet to multiple destinations, the Internet Datagram Protocol permits multicast identifiers to appear only in the destination network address of an internet packet.

In general, the three fields that define a network address of an internet packet can take on values from one of three different classes: *unknown, all,* and *specific.* The numeric value 0 has been reserved for *unknown,* "all ones" for *all,* and any other value for *specific.* This convention will be used throughout the Internet Datagram Protocol. A value from each of the classes may not be appropriate for each of the three fields of a network address. Figure 3.2 summarizes the conventions described below.

For network addresses containing physical host numbers, the network number can be *unknown* or *specific,* while the host and socket numbers must be *specific.* A host number cannot have the value *unknown.* A network number of *unknown* implies that the packet should be transmitted on the locally connected network(s). The network number *unknown* is a synonym for the network(s) to which the host is connected until the host discovers the *specific* values. The notion of directing a packet to all sockets within a machine has not proved to be very useful, and is disallowed.

For multicast network addresses, the network number can be *unknown, specific* or *all*; the host number is *specific* or *all*; and the socket number is *specific.* The implications of different types of network numbers in these network addresses are examined below.

A multicast group, by its very definition at the internetwork level, can span many individual networks. *Directed multicast* is defined to be the delivery of an internet packet to the members of the multicast group on the network contained in the destination multicast network address. The network number in such directed multicast internet packets is *unknown* or *specific.* A network number of *unknown* implies that the packet should be transmitted on the locally connected network(s). The network number *unknown* is a synonym for the network(s) to which the host is connected until the host discovers the *specific* values. Analogously, *directed broadcast* is the delivery of an internet packet to all machines on the specified network.

Global multicast or *global broadcast* is the delivery of the internet packet to all members of the group within the entire internet. The Internet Datagram Protocol does not support global multicast or broadcast, but this and other forms of multicast (such as on one-hop-away networks) may be implemented by the client as a series of directed multicasts.

It is mandatory for host an an Ethernet to be able to receive broadcast packets. It is also mandatory for the internet to deliver a directed broadcast packet to the target network if it is an Ethernet. Multicast is a generalization of broadcast and is optional.

Supporting multicast has been confined to level one. At this level, the software guarantees only with high probability to deliver a packet to its destination(s). Clients may build their own level two reliable multicast protocols.

The details of the store-and-forward routing algorithms are examined in Section 4 in conjunction with the metrics by which routing is to be performed and the routing tables maintained.

	Network Number	Host Number	Socket Number
Physical	specific/unknown	specific	specific
Multicast — directed	specific/unknown	specific	specific
Multicast — global (unsupported)	all	specific	specific
Broadcast — directed	specific/unknown	all	specific
Broadcast — global (unsupported)	all	all	specific

Figure 3.2 Combinations of network, host, and socket numbers in network addresses

3.4 Packet format

We now describe the various fields in an internet packet.

Checksum

The *Checksum* is an optional ones complement add-and-left-cycle (rotate) of all the 16-bit words of the internet packet excluding the checksum word itself. The checksum described here is a software checksum, and is in addition to any error checking mechanism supplied by the transmission medium. (These mechanisms are usually specialized to detect the specific sorts of errors commonly encountered on particular transmission media). As a software checksum, it is designed to be easily computable on Network Systems processors, to be incrementally updatable if the software desires to change only a few words of the message, and to provide reasonable error detection. It is assumed that one of the functions of this checksum is a

verification of the path between the communication hardware and memory. Therefore, it is unreasonable to compute the checksum as the data flows along that path.

When computing the checksum on an internet packet whose length is an odd number of 8-bit *bytes*, there will be included in the checksum a garbage byte at the end of the data (see description of Length field below). *This byte is not presumed to be zero, and must be preserved in the transmission of the internet packet.* Internet packets are, therefore, always transmitted as an integral number of 16-bit words.

Sources of internet packets that do not want to generate a checksum and are willing to tolerate a reduced transmission reliability may make the checksum field be a -1 (two's complement), which is the same as the ones complement "minus zero" (177777 octal). This value indicates that no checksum is present. If the result of the checksumming operation is the one's complement value "minus zero" (177777 octal), it should be converted to "plus zero" (0 octal). The value 177777 octal is *specifically* defined to mean that the internet packet is unchecksummed.

The internet packet's checksum is carried with it from source to destination. If and when an internet packet is altered en route, for example, when the router Hop Count in the Transport Control field (described a little later) is incremented, the checksum must be recomputed. The choice of the ones complement add-and-cycle is intended to permit incremental checksum updating. The algorithm for updating the Checksum after changing a single word of the internet packet is as follows (ones complement arithmetic used throughout):

(1) If the checksum field is 177777 octal, do nothing.

(2) Otherwise, subtract the old contents of the changed word from the new contents using one's complement arithmetic.

(3) Left-cycle (rotate) this difference (n mod 16) bits, where n is the distance (in 16-bit words) from the changed word to the end of the internet packet. (The distance between two words in memory is the difference in their addresses plus one.)

(4) Add the result to the existing Checksum field.

The above procedure produces a correct checksum if and only if the original checksum was correct.

Length

The *Length* field carries the complete length of the internet packet measured in bytes, including the Checksum. This is the length of the Data plus 30 bytes, where the Data includes all of the level two protocol information. An internet packet is *always* carried in an integral number of 16-bit words. The number of 16-bit words is calculated by adding 1 to the Length and dividing by 2. When there are an odd number of bytes in the Data of the internet packet, an extra Garbage Byte is added to fill the internet packet out to an integral number of 16-bit words. This Garbage Byte is not included in the length field.

The Internet Transport Protocols specify a nominal maximum length of 576 bytes for an internet packet. This allows 30 bytes of Header; 12 bytes of Sequenced Packet Protocol overhead (a level two protocol); a typical disk page, which is 512 bytes; and 22 bytes for use by the level three protocol.

Each system element is expected to be able to receive packets of this size. Internetwork routers must at least forward internet packets of this size. *Intranetwork fragmentation and reassembly* must be performed by the level zero protocol for transmission systems that cannot transmit a correct length internet packet [14]. Two communicating clients are free to negotiate a different maximum length.

If the processes negotiate an increase in the maximum length of an internet packet, consideration must be given to the possible problems of any internetwork routers that may be in the communication path. These routers are not a party to the negotiations, and in fact cannot be, since the route is dynamic. If a router encounters a packet which it cannot handle, it will discard it. There is no automatic way to determine whether the routers along the way will or will not support packets of a particular size.

Transport Control

The *Transport Control* field is used to manage the transport of internet packets in the internet. This field is used only by internetwork routers, and should be initialized to zero by the source client process. As internetwork routers change this field, they will incrementally update the Checksum. The *Hop Count* (bits 4-7) is used to count the number of internetwork routers encountered during the transport of an internet packet; each internetwork router must increment this field by one. An internet packet which reaches its sixteenth internetwork router will be discarded. An error indication could be transmitted, according to the Error Protocol, to the source of the internet packet where it might be logged (see Section 5). This ensures that internet packets will not be routed indefinitely in a loop, should the transient behavior of the router's adaptive routing algorithm produce such loops. This assumes that the maximum number of networks an internet packet may traverse is 16. Bits 0-3 currently have no use, should be zero, and are reserved.

The *Maximum Packet Lifetime (MPL)* is an estimate of the maximum time any internet packet will remain in the internet governed by the Internet Transport Protocols. The value chosen for MPL is 60 seconds. It is believed that packet lifetimes will be bounded by MPL with high probability for the majority of internet configurations. Therefore, the magnitude of MPL can be used for tuning system performance, as in determining timeouts.

MPL is estimated to be 60 seconds by observing that in the worst case, if each internetwork router hop produced a store-and-forward plus transmission delay of 1 second, then the delay to traverse 15 internetwork routers would be 15 seconds. This was multiplied by 4 to account for rare delays. In the event that an internet has a very large number of low-speed leased lines, packets may be delayed for more than 60 seconds. While it is expected that packets will be bounded by MPL, under some unexpected circumstances some packets may be delayed in the system for a longer time.

The lifetime of a packet is bounded by requiring that all network drivers discard internet packets that have been on the "transmit queue" for longer than 1/10 MPL.

Packet Type

The *Packet Type* field is used to identify the format of the Data field of the internet packet. It is the bridge between levels one and two, since its meaning is a level two function but its location is defined in level one. This document describes in detail several level two protocols distinguished by this field. The assignment of type values to existing level two packets is summarized in Appendix E, along with the procedures for assigning new packet types. The

level one implementation does not interpret this field; it merely delivers an arriving packet to the appropriate socket, the client of which interprets the Packet Type.

Source and Destination Network Addresses

The *Source* and *Destination* fields specify two internet packet network addresses. The source of the internet packet is the place from which the internet packet originated, and the destination is the place(s) to which the internet packet must be delivered. This specification says nothing about which process initiated the conversation, or the general direction of data flow. It deals only with delivery of the packet at hand. Network addresses are sufficient to identify the source and destination(s) of an internet packet within the entire internet.

Data

The *Data* of the internet packet is a sequence of bytes, transparent to the level one protocol. The length of the data sequence is derived by subtracting 30 from the Length, and may take on any value from 0 to the negotiated maximum (nominally 546 bytes).

Garbage Byte

At the end of the Data of an internet packet containing an odd number of bytes there will be a *Garbage Byte*, which is included in the Checksum (as described above), but not in the Length.

3.5 Implementation notes

A client process interfaces to the Internet Datagram Protocol package by acquiring a socket and then transmitting and receiving internet packets on that socket. This interface imposes certain constraints in order to enforce (inter)network access control, and provides some defaults in order to permit the client process to be oblivious to some of the (inter)network structures and conventions.

The internet will give its best efforts to the delivery of an internet packet, but it *must* be assumed that internet packets will sometimes be lost (even when they are "known" to traverse only transmission media that are believed to be "perfect"). If an internet packet's checksum is checked en route and found to be in error, the internet packet may be thrown away without even so much as a trouble report. Internet packets may also be discarded in the event of a buffer shortage or other resource limitation at any of the places it may pass. The Internet Datagram Protocol does not retransmit internet packets on behalf of its client. It is considered good practice to have the Internet Datagram Protocol package use the Error Protocol via the *well-known router error socket* (see Appendix D) to notify the source of an internet packet's demise, but no process should depend on receiving such negative acknowledgments for all (or indeed any) lost packets (see Section 5). Error Protocol packets are generated in response to broadcast or multicast packets *only* in internetwork routers, if at all.

The destination address is specified by the source client process. The source socket is provided and/or verified by the source host's interface to this level. The source host and source network are specified and/or verified by the Internet Datagram Protocol package.

If a host is connected to many networks, then an active socket within the host will have many equivalent network addresses, each one differing in the network field. The internetwork

routing algorithm (see Section 4) treats the network number in the destination address of an internet packet as a partial route by which to deliver the packet. Therefore, whenever a service advertizes its network address it must take care to advertize all of them, and conversely if a consumer of a service cannot reach the service at one network address it must try the others before giving up [1; 10].

The Internet Datagram Protocol package implements a router that makes use of a routing table by which an internet packet is directed towards another (internetwork) router on its way to the final destination. The routing table may be updated dynamically using the Routing Information Protocol described in Section 4.

Care must be taken when initiating a directed broadcast or multicast, since such packets potentially affect many hosts. A router that gets a broadcast or multicast internet packet processes it according to the following rules:

(1) If the internet packet originated from a socket (in the same host as the router), then it is forwarded to the (internetwork) router that is connected to the directed network. Once there, the router hands the packet to the appropriate Transmission Medium Protocol package for encapsulation and transmission if the network supports broadcast or multicast (see Section 2). If the internetwork router discards the internet packet because the destination network does not support broadcast or multicast, then it notifies the source of the packet via the Error Protocol (see Section 5).

(2) If the internet packet comes from the (broadcast or multicast) network identified in the destination multicast network address, then the router delivers the internet packet to a socket that is in the same machine as the router. If no such socket exists, the packet is discarded, and no error is to be generated.

Multicast and broadcast require that all recipients in a group receive the packet on the same socket identifier. Mechanisms for informing multicast group members values of well-known multicast identifiers and socket numbers are outside the scope of this document.

Multidestination delivery of an internet packet can be achieved by broadcasting the packet to a well-known socket. This is very inefficient if the number of machines on a network that really want to see the packet is very small, since all other hosts must process the packet, only to discover that they do not want it.

The specification of a level two protocol should use a minimum number of packet types in the internet packet to identify its level two packet formats. The preferred style is to use one packet type at level one, and various "commands" or "operations" at level two. The Routing Information Protocol and Echo Protocol (see Sections 4 and 6) are examples of the preferred style. In addition, the specification of a level two protocol should not be based on the source socket number That is, the level two packet should be parsable without using the source socket number as a guide; packet types, operations, commands, etc. should be sufficient. Socket numbers (even well-known ones) are addresses, and address-based parsing of packets leads to special cases that may interfere with alternative binding schemes, and the protocol architecture on the whole.

4

Level two: Routing information protocol

4.1 Introduction

The routing of internet packets as datagrams through the internet is performed using a store-and-forward algorithm that makes use of a routing table in the router of *each* system element. This database directs packets toward another (internetwork) router on its way to the final destination. The *Routing Information Protocol* is the means by which this database is dynamically maintained, and therefore, the means by which routers locate internetwork routers to other networks. This is a level two protocol.

All routers implement a *routing information process* that, in addition to being a *requestor* (the consumer) of information, may also be a *supplier* of information. By convention, this process uses the *well-known routing information socket*, whose value is specified in Appendix D. Routing Information Protocol packets will generally be transmitted and received at this socket. The routing information supplier responds to requests for information, and may also distribute routing information gratuitously. The routing information requestor uses the routing information to maintain a *routing table cache*. A requestor exists in all routers, while a supplier exists only in *all* internetwork routers. Internetwork routers are special routers as they spend most of their time routing packets among networks and supplying routing information to other routers; but they, too, have a routing information requestor (see below).

Routing information suppliers must know the identity of the networks to which they are connected. Routers that implement only the routing information requestor may use the Routing Information Protocol to discover the identity of the network(s) to which they are connected.

The design of the Routing Information Protocol, and the algorithms associated with routing table maintenance are based on the semantics associated with network numbers in internet packets, and the metrics by which a route is picked.

4.2 Assumptions on current internetwork topologies

The fundamental assumption for routing in the internet is that the internet contains at most a few hundred networks. As a consequence, it is sufficient for the network number space to be *flat*, and for all internetwork routers to maintain a complete image of the internet in their routing tables. We currently do not use area- or hierarchical-routing.

We assume that an internet may contain Ethernets, leased lines, and public or private data networks that use packet or circuit switching. Even though these networks have different delay-bandwidth characteristics, we define the metric for internetwork delay to be *internetwork router hops.* Internetwork routers use the Routing Information Protocol to inform one another of the topology of the internet. We expect the predominant network in the internet to be the Ethernet, and in this configuration the Routing Information Protocol makes use of the broadcast capabilities of the Internet Datagram Protocol and Ethernet. Public data networks, however, may not support broadcast, or broadcast may be expensive; in these configurations, internetwork routers must know the identity of the other internetwork routers with which they periodically exchange routing information. The frequency with which routing (and topology) information is exchanged is discussed later.

The format of the Routing Information Protocol packet is shown in Figure 4.1 and described in Section 4.3. The packet type for such internet packets is *routing information protocol,* which is specified in Appendix E.

It is recognized that the current assumptions and, therefore, the protocol for updating routing tables will not be suitable in large complex internetwork topologies for a number of reasons. First, it is inefficient and time consuming for all internetwork routers to hold and update an image of the entire internet. We expect there to be natural localities of activity and interaction among the hosts, and expect that for the most part a host will communicate with other hosts in the same "neighborhood". Second, large internets will have networks of different delay-bandwidth characteristics. This, in conjunction with queuing delays in internetwork routers, must be reflected in the information exchanged between routers if they are to pick nearly optimal routes. Mismatches in the delay-bandwidth characteristics of the constituent networks may also result in congestion.

4.3 Packet format

Operation

The first word of the level two fields of the Routing Information Protocol packet indicates the *Operation* specified by this packet. The operation can be *request* or *response,* indicating that the packet is a request for routing information or contains routing information, respectively.

The operation field is interpreted as follows:

Operation	Description
1	request
2	response

Contents

The *Contents* of a Routing Information Protocol packet will be one or more tuples, each of which consists of a 32-bit *Object Network* number and a 16-bit *Internetwork Delay* in hops.

If the operation specified in the Routing Information Protocol packet is *request,* then in each tuple the object network is the number of the network in which the requestor is interested, and the internetwork delay is typically set to *infinity. Infinity* is defined to be 16 hops. If the requestor wants information on all object networks known to the internetwork router, then it

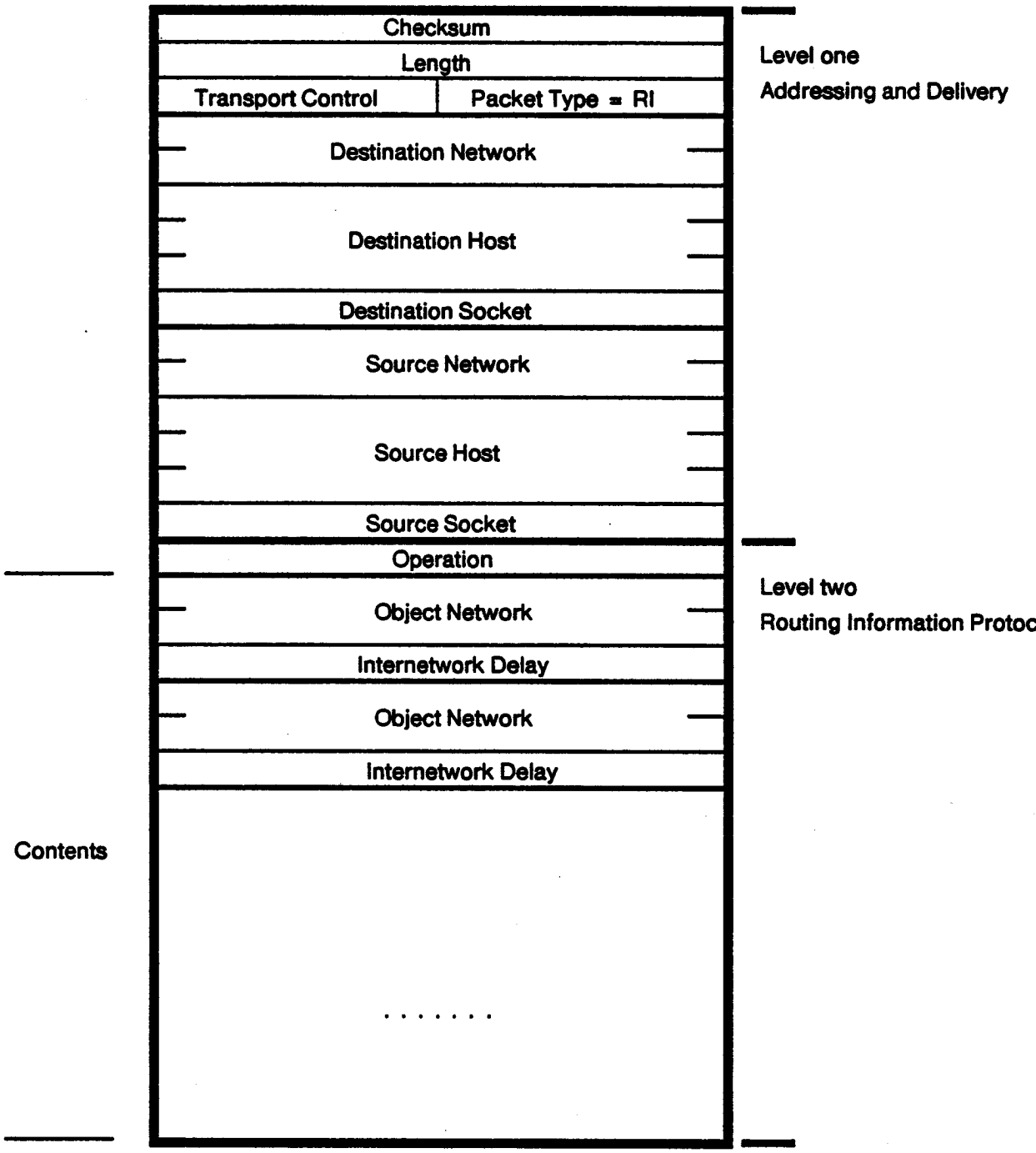


Figure 4.1 The routing information protocol packet

will include only one tuple in which the object network is set to *all* (see Section 3.3) and the internetwork delay is set to *infinity*.

If the operation specified in the Routing Information Protocol packet is *response*, then in each tuple the internetwork delay is the delay to reach any host on the object network via the internetwork router generating the packet.

Routing information supplier processes respond to requests for information, and also periodically broadcast all of their routing information on all directly-connected networks to inform other (internetwork) routers.

When a routing information supplier transmits or broadcasts a packet containing a *response* over a directly-connected network, the semantics of a tuple in the contents are as follows. The internetwork delay indicates the delay experienced by a packet to reach any host on the object network from the directly-connected network via the internetwork router that generated this response packet. In other words, the internetwork delay is an estimate of the delay experienced by a packet from the time it just leaves the "directly-connected network" via the internetwork router, to the time it just enters the object network. The tuple corresponding to a directly-connected network over which the packet is being transmitted has an internetwork delay of one. This indicates that there is one internetwork router hop delay to reach the directly-connected network from the same directly-connected network, using the internetwork router. A delay of *infinity* implies that the object network is unreachable, or that the internetwork router does not know about the object network. The delay to reach a router on any directly-connected network is zero hops.

4.4 Routing table maintenance

The Routing Information Protocol is used in each router to maintain a cache containing the routing table. The following algorithms are recommended for maintaining the routing table. (The routing table in the internetwork router can be thought of as a very large cache.) These algorithms serve two purposes:

(1) Permit the quick initialization of the routing table when a host starts running.

(2) Ensure that the routing table remains up-to-date as internetwork topology changes (that is, new internetwork routers come up, or existing ones go down temporarily or permanently).

The routing table at each router contains an entry for each object network in which the router is interested. Each entry contains: (1) a delay, (2) the directly-connected network (one of the potentially many that the host is connected to) on which to transmit the packet, (3) if the delay is not zero, then the host number of an internetwork router, on the directly-connected network, via which packets will reach the object network with that delay, and (4) a timer used in the routing table maintenance algorithm. At initialization time, the routing table in a router which implements only a routing information requestor process contains only an entry for the directly-connected network(s). At internetwork routers the routing table contains an entry for all directly-connected networks. (Recall that internetwork routers *must* know the identity of all directly-connected networks.)

When a Routing Information Protocol packet containing a *response* is received (either in response to the router's *request* or gratuitously via a broadcast from an internet router), the following procedure is used to process each tuple in the contents and update the routing table.

A decision must be made as to whether or not it is appropriate to update the routing table entry corresponding to the object network in the tuple with new information. If the object network is known to be directly-connected (i.e., exists in the local routing table with a delay of zero), then no update is necessary. Otherwise, an update should be performed if any of the following conditions is true:

(1) There exists no routing table entry for the object network and the host would like to create an entry.

(2) The source host of the Routing Information Protocol packet is the same as the address of the internetwork router in the existing routing table entry. That is, updated information is being received from the very internetwork router through which packets for the object network are being routed.

(3) The existing entry in the routing table has not been updated for some time (defined to be 90 seconds), suggesting that it may no longer be valid.

(4) The internetwork delay from the tuple, plus an estimate of the delay between this routing information user and the source of the response packet, is less than the delay in the existing routing table entry; that is, the new information describes a better means of reaching the object network than the existing routing table entry.

If any of the above conditions holds, then the routing table entry is updated as follows:

(1) The host identity of the internetwork router in the entry is set to the source host in the Routing Information Protocol packet.

(2) If the internetwork router is on a different directly-connected network, then the identitiy of the network on which to transmit packets to the object network is updated.

(3) The delay in the entry is set to the internetwork delay. If the internetwork delay exceeds 15 hops, then the object network is deemed to have become inaccessible. The internet delay is equal to the internetwork delay from the tuple, plus the delay to the internetwork router. This is equivalent to the delay from the tuple since the delay to an internetwork router on a directly-connected network is zero.

(4) The entry's timeout is reset to 90 seconds.

In addition to processing Routing Information Protocol packets, the routing information requestor process also periodically checks for routing table entries that have not been updated for so long that they are almost assuredly invalid. The timeout used for this is 3 minutes, or twice the 90-second interval after which the above algorithm deems an entry to be suspect. Entries not updated for 3 minutes have their delay set to infinity, and are candidates for garbage collection should the implementation require it. Internetwork routers typically keep such entires around for another 60 seconds or so to make sure that the entire internet will know that the object network is unreachable. During this time the entry may be updated or processed just as any other entry in the table.

A routing information supplier gratuitously transmits a copy of its routing table (in one or more *response* packets) to other internetwork routers on all networks to which it is connected. The *response* packets are transmitted approximately once every 30 seconds. The packet is either broadcast on the network, or specifically addressed to the internetwork routers if the network does not support broadcast. In addition, whenever an internetwork router modifies the delay field for an entry in its routing table, it immediately informs all internetwork routers on directly connected networks with a *response* packet. This action propagates changes in routing information through the internet very quickly. *Response* packets contain routing information for entries in the supplier's routing table. The delays in the response packet are calculated by adding one internetwork router hop delay to all entries in the routing table (with a maximum of *infinity*). If for any reason the internetwork router is about to be turned off, it gratuitously transmits a *response* packet in which the delay for *all* object networks is *infinity*. This informs the rest of the internet that the internetwork router is unusable well before it actually becomes so.

When a routing information supplier receives a *request* packet, it supplies the appropriate information. If it does not have an entry for a requested network it returns *infinity*, since that network is unreachable.

4.5 Properties of the protocol

The Routing Information Protocol and algorithms described above are adequate for maintaining routing information in an internetwork whose topology changes relatively infrequently and whose constituent networks have a slowly changing transmission delay.

Since most of the networks in Xerox' internets will be Ethernets, the networks have a relatively constant and equal transmission delay. Hence, internetwork delays can be measured in the number of internetwork routers that must be traversed. When an internetwork router comes up, affording a shorter route (or the only route) to some network than any already in existence, all routing tables will have been updated to reflect the new information within approximately $30*n$ seconds, where n is the number of hops from the new internetwork router to the most distant network, and 30 seconds is the routing information server broadcast interval.

When an internetwork router goes down and if another route exists to some network previously accessible via that router, it will be discovered and used (though perhaps not in an optimal fashion) within 90 seconds (the time after which a non-updated routing table entry becomes suspect and eligible for replacement by one that appears less favorable). If a network becomes inaccessible, the entry for that network will be purged from all local routing tables within approximately $90 + 30*\max(n, 15)$ seconds (usually much faster than this). During this interval, it is possible that the routing tables in some internetwork routers will be such that packets destined for the inaccessible network will be routed in a loop. However, this is not a stable situation, since the internet packet's hop count field will be incremented as it is passed from router to router and will quickly exceed the maximum value of 15.

The responsiveness of the system is actually a little better than described above, since internetwork routers send out messages whenever delays in routing table entries change.

This routing table update algorithm is stable, provides adequate responsiveness, and does not require a complicated implementation.

4.6 Implementation notes

The destination host number in a *request* packet will be *specific* if the router knows the identity of a routing information server process, or will be *all*, thereby causing the packet to be broadcast to all hosts. Similarly, the network number in such packets will be *unknown* if the host does not know it or will be *specific* if it has discovered it by some previous action. The destination host number for a *response* packet is either *all*, indicating that the response is being sent gratuitously to all routing information user processes, or *specific* because the response is targeted back to the host requesting the information.

When a new host comes up, and only has a routing information requestor process, it broadcasts a request packet along all directly connected networks and builds its routing table from the response packets. The routing information requestor process will deduce the identity of its directly connected networks, since response packets will have the correct network number (non-zero) in the source network address field.

Hosts that implement routing information supplier processes, that is, internetwork routers, will know the network number associated with their directly-connected networks. By broadcasting response packets they are able to inform other such supplier processes about the networks they are connected to. Routing tables in internetwork routers therefore are constructed, and knowledge about the collection of networks proliferates through the internet. Internet routers may of course delete entries in their routing tables for those entries that have not been updated within 3 minutes. An entry with a delay corresponding to *infinity* indicates that the object network is inaccessible. Tuples for a particular object network in *response* packets are created by adding a delay of one internetwork router hop to the delays that it has for that object network in its routing table. Gratuitous responses by an internetwork router need not include tuples for object networks it believes are unreachable.

A routing table supplier process may often get requests for information from agents other than routing table user processes. For example, network measurement and monitoring agents periodically or on demand determine the state of the internet.

A number of cache management techniques may be used for implementing the routing table in hosts other than internetwork routers. In specialized applications where it is known that there will be only one active conversation at a time (in simple workstations, for example), the routing table may consist of a single entry corresponding to the network of the other partner in the conversation. (Of course, the router must also keep information about the identity of the directly connected networks.) A further simplification may be employed in very simple servers on the Ethernet that only transmit packets in response to an arriving packet. The response is sent to the (internetwork) router identified as the immediate source in the encapsulation of the arriving request. This takes advantage of the assumption that the reverse of the route taken by the request packet is an acceptable route for the response; it does not require processing Routing Information Protocol packets at all.

4.7 What is a network?

So far, we have used a rather intuitive definition of the term network—a transmission medium configured to carry internet packets according to the conventions of level zero transmission media protocols (see Section 2). This definition may be applied without any modification to the Ethernet, and any other datagram communication system. However, by defining *network* carefully, we can devise rules by which it is possible to incorporate circuit-oriented networks

into the Internet Transport Protocols' datagram internet. The generalization we are about to describe will permit *standalone* system elements to connect to an internet router (through phone lines or other commercially available circuit-oriented communication systems), and become a *remote* system element (with the same priviledges as any other system element in the internet) for the duration of time it is part of the internet.

We classify hosts into two groups: internetwork routers and stations. Stations may be workstations or servers. A *network* is a communication system configured to carry internet packets that both connects stations together and is connected to an internetwork router. Stations on the same network may communicate amongst themselves without the explicit aid of the store-and-forward services of the internetwork router. A network is assigned a network number.

According to this definition a communication system (a circuit or leased line in many cases) that *just* connects internetwork routers is not a network and need not have a network number. Neither would a communication system that *just* connected stations. We specifically require that Ethernets by themselves do in fact get a network number, since the probability that they will remain by themselves is very low.

Circuit-oriented communication systems that connect stations to an internetwork router are not given a network number. Rather, the collection of circuit ports at an internetwork router is identified as a network with a network number. (A circuit port may be a physical device identified by a modem, say, or may be a logical device as in X.25.) Stations on such circuit-oriented communication systems will have the network number corresponding to the internetwork router at which their circuit terminates. The internetwork router at which their circuit terminates may change from session to session without any problems. Standalone workstations should not be assigned a network number of their own. Therefore, we have stretched our definition of network to describe a collection of circuit ports (possibly of different types) at an internetwork router as a network; the circuit handling drivers can be thought of a "switch" connecting stations and the internetwork router together. A station that communicates with another on the same network will send its packets to the switch (resident as a "network driver" on the internetwork router) that will loop the packet back onto another circuit.

Techniques for the initialization and management of the internet have been optimized for datagram, broadcast networks. The internet is capable of incorporating non-broadcast datagram networks, and circuit-oriented communication systems with additional specialized initialization and binding.

5

Level two: Error protocol

5.1 Introduction

The *Error Protocol* provides a means for any agent in the system to report that it has noticed an error (and as a result discarded the packet). The use of the protocol is optional, and it is intended as a diagnostic tool as well as a means to improve performance.

An Error Protocol packet is returned from the destination socket in the host detecting the error. If the Error Protocol packet is generated on behalf of the Internet Datagram Protocol package (by the router or internetwork router), then the source socket of this packet will be the *well-known router error socket*, which is specified in Appendix D. The packet is always destined to the source socket of the packet that provoked the error. An Error Protocol packet is never generated in response to the receipt of an Error Protocol packet (to avoid infinite loops). In addition, an Error Protocol packet is never generated in response to a multicast or broadcast packet (see Section 5.3). The format of the Error Protocol packet is shown in Figure 5.1 and described in detail below. The packet type of such internet packets is *error protocol*, which is specified in Appendix E.

5.2 Packet format

The *Contents* of the Error Protocol packet at level two is two 16-bit words, indicating the *Error Number* and *Error Parameter* respectively, followed by the first portion of the offending packet. This portion must include all of the Internet Datagram Protocol fields (level one), and as much of the level two and higher fields as the implementor chooses to copy. It is recommended that at least 42 bytes be copied, since that accommodates level two of the Sequenced Packet Protocol. Since the Error Protocol packet contains a portion of the offending packet, the receiver of Error Protocol packets has information validating its authenticity.

The Error Number indicates the kind of error, and the Error Parameter is a parameter for some kinds of error. If a specific error parameter does not exist, then the Error Parameter should be set to zero:

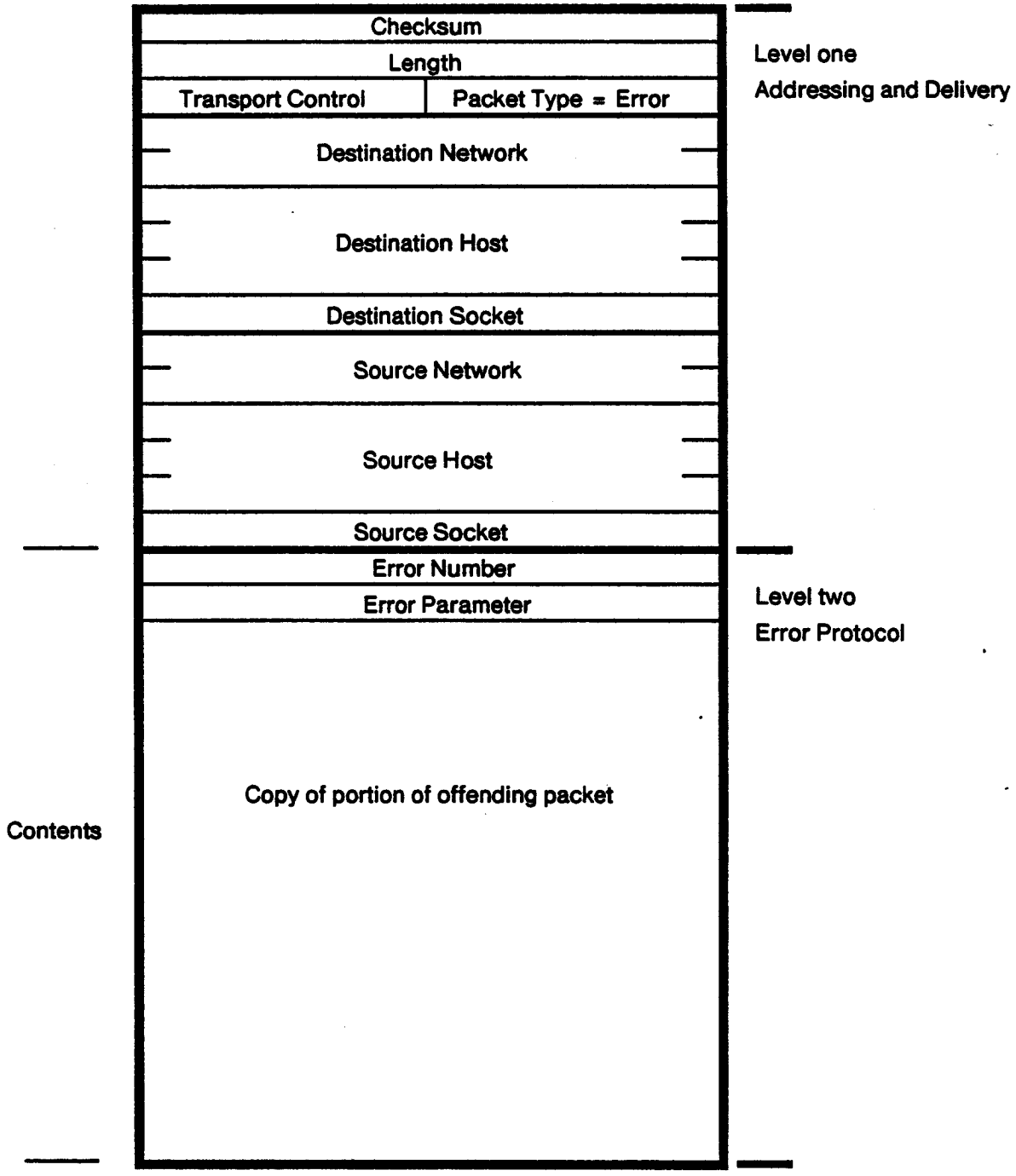


Figure 5.1 The error protocol packet

Error Number (octal)	Description
0	An unspecified error is detected at destination.
1	The checksum is incorrect, or the packet has some other serious inconsistency detected at destination.
2	The specified socket does not exist at the specified destination host.
3	The destination cannot accept the packet due to resource limitations.
1000	An unspecified error occurred before reaching destination.
1001	The checksum is incorrect, or the packet has some other serious inconsistency before reaching destination.
1002	The destination host cannot be reached from here.
1003	The packet has passed through 15 internet routers without reaching its destination.
1004	The packet is too large to be forwarded through some intermediate network. The Error Parameter field contains the length of the largest packet that can be accommodated.

5.3 Implementation notes

The software entity receiving an Error Protocol packet—at a socket it accesses—processes the packet in whatever way it deems suitable. For example, software implementing the Sequenced Packet Protocol may receive an Error Protocol packet indicating that "the specified socket does not exist at the specified destination host". If the connection was previously open, then the software can conclude that the process at the remote machine must have "deleted" the connection and gone away. It is recommended that at least 42 bytes (if not more) of the offending packet be copied into the Error Protocol packet, so that the software entity processing such a packet can return an appropriate response to its client.

The Error Protocol is, thus, implemented in conjunction with other level two protocols. It has been identified as a separate protocol so that its semantics may be applicable to *all* level two protocols that wish to use it. It is recommended that implementors of level two protocols generate and process Error Protocol packets as and when necessary.

An Error Protocol packet is not generated in response to a broadcast or multicast packet in routers except internetwork routers that are forwarding a directed broadcast or multicast. This prevents the generation of a large number of unnecessary Error Protocol packets by hosts that do not wish to process the broadcast or multicast packet.

6

Level two: Echo protocol

6.1 Introduction

The *Echo Protocol* is used to verify the existence and correct operation of a host (and the path to it). This simple protocol specifies that all Echo Protocol packets received should be returned to the source. The format of an Echo Protocol packet is illustrated in Figure 6.1 and described in detail below. The packet type of such internet packets is *echo protocol*, which is specified in Appendix E.

6.2 Packet format

The first word of the level two portion of the Echo Protocol packet contains an *Operation*, indicating whether the packet is an *echo request* or *echo reply*. The reply will be checksummed if the incoming request was checksummed. If the packet is received in error, then the reply should be according to the Error Protocol and not the Echo Protocol. The echoed data will consist of the data in the arriving packet. Therefore, the receiver of the echoed packet has information validating is authenticity.

The Operation field is interpreted as follows:

Operation	Description
1	echo request
2	echo reply

6.3 Implementation notes

An *echoer* obeying the Echo Protocol, by convention, should be implemented on the *well-known echoer socket* as specified in Appendix D at every host. Echo Protocol packets may be sent to any socket, and will be echoed if the client at that socket is willing to implement the Echo Protocol. The Echo protocol can be used to verify the correct operation of an implementation of the Internet Datagram Protocol.

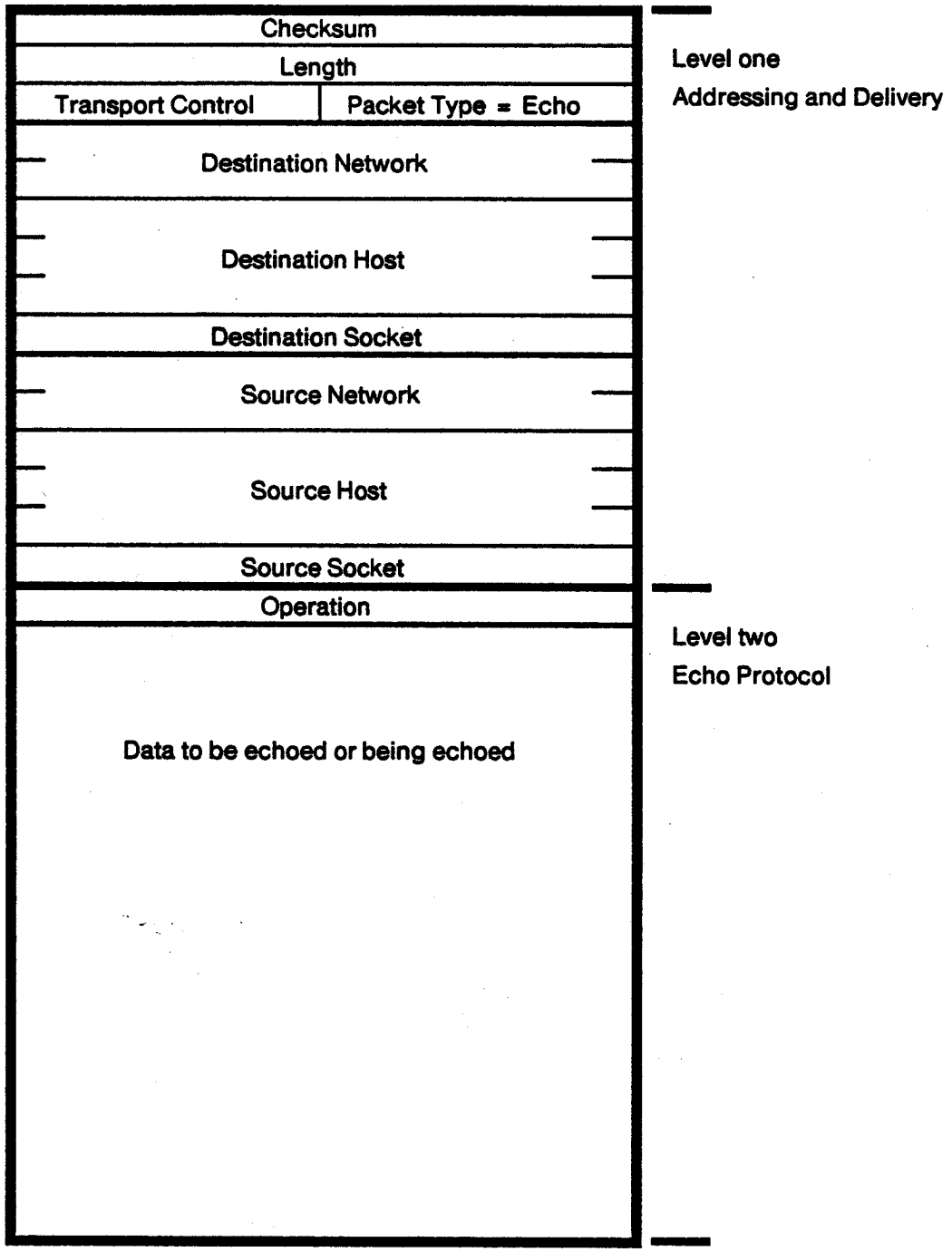


Figure 6.1 The echo protocol packet

7

Level two: Sequenced packet protocol

7.1 Introduction

The *Sequenced Packet Protocol* provides reliable transmission of successive internet packets on behalf of client processes. Each of the client's packets is given a sequence number when it is transmitted by the source. These sequence numbers are used to order the packets, to detect and suppress duplicates and, when returned to the source, to acknowledge receipt of the packets. The flow of data from the sender to the receiver is controlled on a packet basis. The source may transmit packets having sequence numbers up to and including the number specified by the receiver.

A *connection* is a transient association of two sockets during which the sequence numbers on transmitted packets are synchronized. When a connection between two sockets is first opened, both ends commence transmitting packets with sequence number zero.

It is important to insure that the failure of one of the partners in a connection is noticed by the other partner. The protocol includes two 16-bit *connection identifiers*, one specified by each end of the connection. Each end tries to insure that a failure followed by a restart of the same program will never result in reuse of the same identifier, so that the restarted program will be easily distinguished from the old instance of the same program. In other words, the two incarnations of the connection will be distinguishable. One way to implement this is to select the connection identifier from a clock register of the processor.

This protocol encourages the maximum size of a packet to be 576 bytes, which is the nominal standard internet packet size. Client processes may negotiate use of a different maximum size when opening a connection. The mechanisms for negotiating changes in maximum packet size lie outside the domain of the Sequenced Packet Protocol.

In an acknowledging protocol such as this one, the recipient of a packet has a somewhat difficult decision to make in deciding whether to acknowledge the packet or to wait and acknowledge several at once. In order to facilitate this decision, the source is given the option of specifying that it wants an immediate acknowledgement.

The specification of this protocol includes the format of the packet, the meaning of sequences of packets, and recommendations on how the protocol should be used to achieve the desired results. The format of the Sequenced Packet Protocol packet is shown in Figure 7.1. The

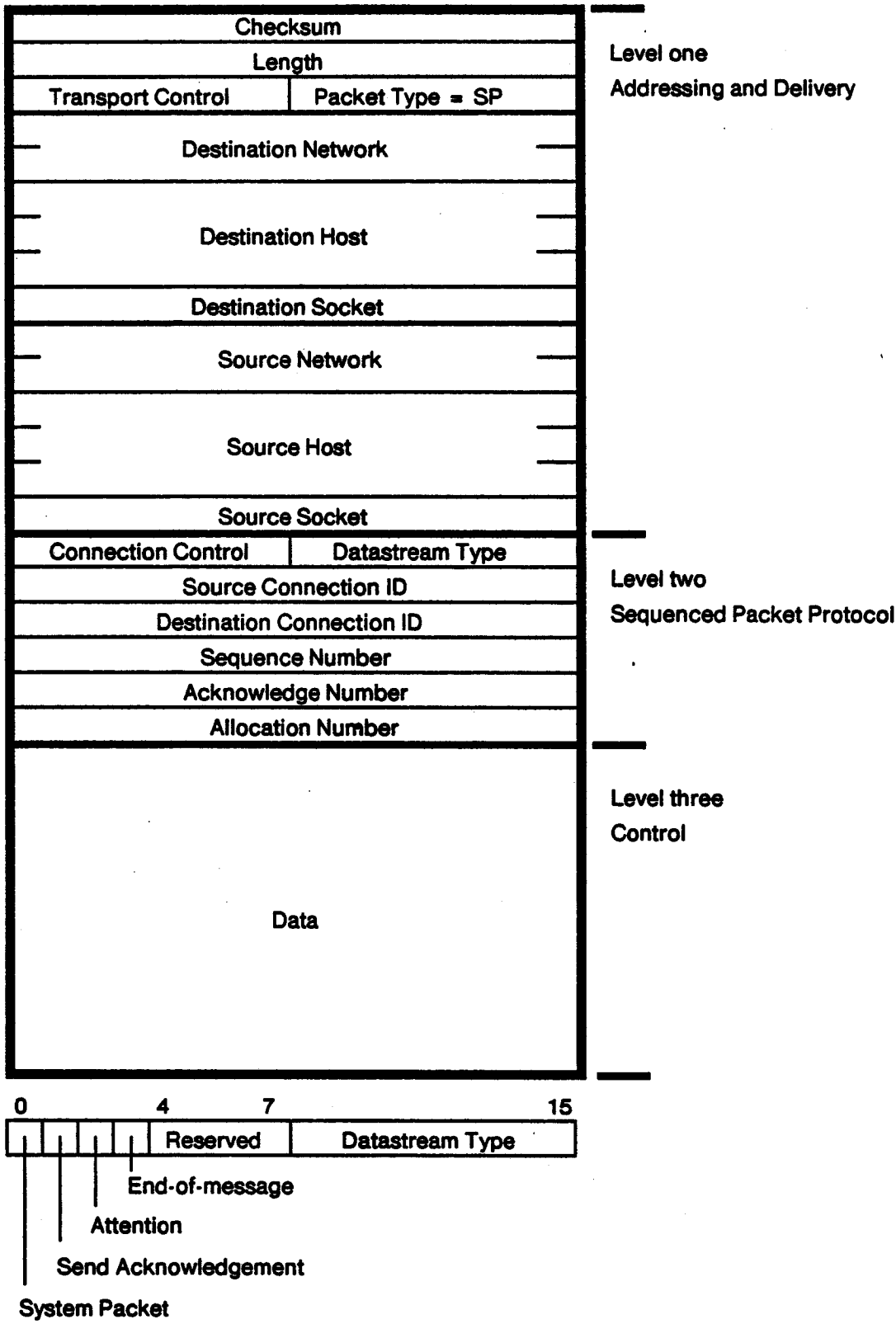


Figure 7.1 The sequenced packet protocol packet

packet type for such internet packets is *sequenced packet protocol*, which is specified in Appendix E.

7.2 Packet format

Connection Control

The *Connection Control* field contains four bits (Bits 0-3) that control the actions of the Sequenced Packet Protocol. The remaining 4 bits are as yet unassigned and should be zero. The assigned bits are labelled *System Packet*, *Send Acknowledgement*, *Attention*, and *End-of-message*, from left to right.

The Sequenced Packet Protocol transports packets reliably between two sockets. It is assumed that the client processes wish to see their packets sequenced contiguously. In performing the necessary functions of sequencing, duplicate suppression, and error recovery, the Sequenced Packet Protocol package at the source may often have to transmit packets to its counterpart at the destination even when there is no client-initiated transmission of packets. Such packets have the System Packet bit set, carry no data, and are assigned (but do not consume) the next unused sequence number. These packets are never seen by the client processes. The System Packet bit enables the software to decide if a packet is for a client process or not. The software otherwise has no way of determining this fact, since the semantics of the Datastream Type (see discussion below) are transparent to the Sequenced Packet Protocol. The System Packet bit may be used to send *probes* (see Section 7.7), or return acknowledgements.

If the Send Acknowledgement bit is set, the source desires that the receiver should send back an acknowledgement. If there is client-initiated traffic in the opposite direction, then acknowledgements will arrive in them, otherwise the protocol package at the remote end will generate a system packet that contains an acknowledgement (see Section 7.7). Of course, a system packet containing an acknowledgement can always be sent back gratuitously without producing any undesirable effects (see Section 7.7).

If the Attention bit is set, the source client process desires that immediate notification be given to the destination client process that this packet has arrived. The packet will eventually work its way through the queue of pending packets according to its sequence number and be delivered *again* in the usual way. If the Attention bit is set, the allocation mechanism described below will be ignored and the packet sent, despite the fact that the destination has no room. The destination is expected to take heroic action to try to accept such packets; the source is expected to send them infrequently. For this reason only one byte of data must be present in these packets. The System Packet and Attention bits *cannot* both be set in a packet.

Clients of the Sequenced Packet Protocol must associate no semantics with packet boundaries, but they may view the stream of packets as a stream of *messages*, if they wish, and the protocol provides a mechanism to punctuate the stream of packets with end-of-message boundaries. If the End-of-message bit is set, then the packet and its contents terminate a message and the next packet will begin the following message. The Sequenced Packet Protocol associates no semantics with messages, other than conveying their boundaries to receiving clients in an efficient manner. This function may be viewed as a bridge between this level of protocol and the next higher layer, very much like the Datastream Type field to be described next. The System Packet and the End-of-message bit *cannot* both be set in a packet at the same time.

Datastream Type

The *Datastream Type* field is the bridge between this level two protocol and level three protocols. This field is ignored by the Sequenced Packet Protocol. The purpose of the field is to provide a convenient way for higher-level protocols to send control information without providing escape machinery in their normal data flow. For example, it provides a means to abort the sending of a very long file, or to mark the end of a file without knowing its length in advance. This is accomplished by putting the Datastream Type in a special place in the packet, so that it may easily be recognized.

The scenario of a typical higher-level conversation might involve a short series of interactions of Datastream Types that are commands, followed by a long, one-way series of packets of Datastream Type "data", followed by a packet of Datastream Type "end-of-data", and responded to by a packet of Datastream Type "data-received-and-processed". After several such sequences, one end or the other might issue a Datastream Type "time-to-quit" packet, which would be followed by the destruction of the connection at both ends. A packet with the Datastream Type "interrupt" in conjunction with the Attention bit set might be used if the transmission seemed to be bogged down, and it might originate at either end.

The Sequenced Packet Protocol deliberately specifies no means for terminating a connection. The decision to stop rests entirely at higher levels. The protocol does recommend that client processes terminate the connection appropriately if they wish the data transmitted to be correctly received and processed. An example of a general way by which connections can be terminated, using packets with suitable Datastream Types, is described in Section 7.5.

Source and Destination Connection Identifier (ID)

These two fields are safeguards against the inadvertent reuse of socket numbers in a destructive way. One *Connection ID* is specified by each end of the connection at the time of its creation, and the connection IDs are checked on all subsequent transmissions. The particular problems addressed by this mechanism are: (1) the sudden restart of a machine, which might come up in the middle of a conversation in such a way that the other end would not know that such a drastic thing had happened, and (2) the reuse of connections between a pair of sockets in rapid succession such that packets from previous incarnations that are delayed in the internet could be delivered by mistake.

New incarnations of a connection would have a different pair of Connection IDs. If the Connection ID is specified by the reading of a clock, or some other random mechanism, then the probability of such a failure has been drastically reduced. A packet should be delivered to the client process only if the Connection ID pair in the packet matches that for the connection. More will be said later about the mechanism for picking Connection IDs and making them known to the other end.

Sequence Number

The *Sequence Number* counts packets sent on the connection. Each direction of data flow is independently sequenced. The first packet is assigned number zero, and the count proceeds from there. If the count overflows the 16-bit field, the overflow is ignored and the count proceeds from zero again. The destination will use the sequence number to deduce the order of packets, to acknowledge them, to suppress duplicates, and to specify flow control information. Only the client process' packets will consume sequence numbers. Packets with the System Packet bit set will be assigned the next available sequence number but will not

consume that sequence number. Note that it is unnecessary to reliably transmit a system packet. One must only know its position in the sequenced flow of packets. Any control information, such as Attention (and possibly Send Acknowledgement) will always be duplicated in a packet that consumes the sequence number assigned to the system packet, and hence will be reliably delivered.

The sending end of the protocol package will expect to receive an allocation indicating that it may use sequence numbers up to and including a particular value (see the Allocation Number field described below). Allocations should rarely exceed a few packets in normal usage. The receiving end should expect to see packets carrying sequence numbers starting from the last one received and ending with the last one allocated. The receiving end may also see packets with sequence numbers one or two beyond the allocation, as for example with attention packets or a probing sender, and sequence numbers up to a few thousand before the last one received, as for example with duplicates. The arrival of packets with sequence numbers outside the reasonable limits is grounds for generating responses using the Error Protocol (see Section 5).

Acknowledge Number

The *Acknowledge Number* field specifies the sequence number of the first packet which has not yet been seen travelling in the reverse direction. That is, the Acknowledge Number indicates the sequence number of the next expected packet. The acknowledging protocol package takes all responsibility for the packets, even if they have not been delivered to the destination client process. The protocol package can, however, not guarantee that the client process will, in fact, read the data. Acknowledgements are cumulative; by specifying the sequence number of the first packet not yet seen, all packets with sequence numbers preceding this one have been acknowledged.

Allocation Number

The *Allocation Number* field specifies the sequence number up to and including which packets will be accepted from the other end. Said another way, one plus the difference between the Allocation Number and the Acknowledge Number indicates the number of packets that may be outstanding in the reverse direction. Buffering will be provided for packets of negotiated (or nominal) maximum size. As described above, the allocation mechanism is ignored for packets with the Attention bit set by both ends of the connection. Once specified, an allocation cannot be revoked, that is, Allocation Numbers in packets are always monotonically increasing. Note that an Allocation Number cannot exceed the Acknowledege Number by more than 2^{15}, that is, half the sequence number space. If this restriction was not imposed, then it would be possible to accept a duplicate packet in the event that the acknowledgement travelling in the reverse direction was lost.

Data

Data may exist in packets that do not have the System Packet bit set. The Data is a sequence of bytes, transparent to the level two protocol. The length of the data sequence is derived by subtracting 42 from the internet packet Length, and may take on any value from 0 to the negotiated maximum (nominally 534 bytes). Packets with the Attention bit set must have only one byte of data.

7.3 Client interfaces

It is anticipated that there will be at least two different kinds of interfaces to a Sequenced Packet Protocol package—a *packet stream* interface, and a *byte stream* interface. Clients of the Sequenced Packet Protocol on a host need not know the interface used by clients of the protocol on another host; we expect packet stream clients to communicate with byte stream clients.

With the packet stream interface, client processes expect to receive packets in ordered sequence. The client process is aware of packet boundaries (even though they assign no semantics to them) and, in general, the format of the packet. There must exist some convenient way for the client process to be told about: (1) the arrival of a packet with the Attention bit, and (2) the occurrence of some malfunction which warrants aborting the connection. In addition to sending packets, the client process must be provided a way to send an attention packet.

With the byte stream interface, client processes will not see packet boundaries, but only bytes (or blocks of bytes), in ordered sequence. The client process needs to be told about special events, like: (1) end-of-message boundaries, (2) the change in type of the data stream, (3) the arrival of a byte of data in an attention packet, (4) the occurrance of some malfunction that warrants aborting the connection. In addition to sending and receiving data, the client must be provided a way to: (1) mark end-of-message boundaries, (2) change the type of the data stream, and (3) send attentions. One would expect a byte stream interface implementation to be merely a front end to a packet stream interface, but this is a matter of implementation.

The Sequenced Packet Protocol does not preclude the existence of more than one connection to a local socket. The decision to support more than one connection per socket is an implementation issue, although it is recommended that only one connection exist. By having only one connection, no generality is lost and the problem of demultiplexing packets as they arrive at a socket is considerably reduced, especially when it is possible for packets of different protocols to arrive at a socket, as in the case of Error Protocol Packets (see Section 5). Two implementations that have different conventions can still communicate, although the one that supports multiple connections at a socket must always take care not to permit more than one connection to a local socket from the same remote socket.

The Sequenced Packet Protocol can also support another interface or mode called the *reliable packet mode.* This is very much like the packet stream interface except that the client process will expect to receive packets out of order *as they arrive,* but without duplicates. There will be no use for the Attention bit, although the attention can still be used to perform the vital function of bypassing the allocation mechanism. The reliable packet mode implementation is used when the higher-level processes are in a much better position to control the management of storage, and do not want to tie up valuable space for buffers with packets which have arrived out of order. The cost of a reliable packet mode is a somewhat more complex bookkeeping mechanism in the Sequenced Packet Protocol package, which must now keep track of packets passed to the client process out of order. A reliable packet mode also provides a better average response time for packet delivery, which is sometimes of interest for real time transmission of data, such as speech. It is possible for clients of a byte stream interface and a reliable packet mode interface to exchange data.

7.4 Establishing and opening connections

The discussion of the Sequenced Packet Protocol up to now has assumed that: (1) the client processes at each end of the connection somehow knew the host and socket number of the other end, and (2) the protocol package at each end knew the Connection ID of the other end. This is not the normal case, but the protocol already described can easily be extended to handle the case where these parameters are unknown. The mechanism for this requires only an expanded interface between the Internet Transport Protocols package and its local client process, and some well publicized mechanisms within the Sequenced Packet Protocol package. This discussion will assume, without loss of generality, that the client processes at both ends use the packet stream interface.

One end of a connection is said to be *established* when it knows the address (host and socket number) and Connection ID of both ends of the connection. An end of a connection is said to be *unestablished* only if the Connection ID of the remote end is *unknown*. If both ends of a connection are established symmetrically, then the connection is said to be *open*. Data can only flow reliably on an open connection, that is, a packet will be delivered to the client process only if its source and destination Host Number, Socket Number and Connection ID match that associated with the connection.

7.4.1 The general case

Consider, first, the case where two client processes that know each others' addresses wish to communicate. If each client process had somehow managed to determine the Connection ID to be used by the other end, then both ends of the connection are established and the connection is open. The determination of the complete address and Connection IDs could be achieved through some private protocol. The Sequenced Packet Protocol package, however, provides a way for either end of the connection to become established by the very act of receiving a Sequenced Packet Protocol packet on the socket associated with the connection. The following discussion assumes that the Sequenced Packet Protocol package permits only one connection to a local socket. If more than one connection were being supported, the algorithm to cause an end to become established would be slightly different.

A Sequenced Packet Protocol packet arriving on an unestablished end of a connection causes that end to become established to the source address *and* source Connection ID of the arriving packet, if and only if the destination Connection ID in the packet does not mismatch the source Connection ID of that end of the connection. If the destination Connection ID in the arriving packet is *unknown*, this does not constitute a mismatch.

Under the assumption that the communication system does not produce delayed duplicates, this simple mechanism is sufficient to open connections between two processes that know each others' addresses but not Connection IDs, as well as the case where a consumer wishes to obtain service from a server at a known socket. The opening of connections in these two cases, and the protection against delayed duplicates, is described below.

Assume that two client processes which only know each others addresses wish to communicate. Both ends of the connection are unestablished. Assume that there is an initiator of the communication and a listener. Upon receiving a packet from the initiator (with sequence number zero and the initiator's Connection ID), the listener's end of the connection becomes established. The listener may transmit a packet of its own (with both the initiator's and listener's connection ID), causing the initiator's end of the connection to become established.

The connection is now established at both ends and will become open only if the packet received by the listener was genuine, that is, not a delayed duplicate. If the initiating packet was a delayed duplicate, both ends of the connection would not be established symmetrically and packets would not be accepted by either end. Once an end of a connection is established, the Sequenced Packet Protocol package is expected to reject packets from an unacceptable source (Connection ID or address violation), presumably without notifying the local client. Irrecoverable damage may have been done, however, if the listener acted upon any data contained in the initiating packet. The Sequenced Packet Protocol package at the responding end should really not deliver such a packet to the client process since the packet did not contain two Connection IDs. In order to enforce that the establishing packet of a connection not be delivered to a client process, the packet stream interface could send a system packet to establish the other end. Alternatively, the responding end could wait for the retransmission of the packet (with the same data) with two Connection IDs, or it could wait to see the next sequenced packet with two Connection IDs and then deduce that the first one was valid and therefore deliverable.

Instead of having an initiator and a listener, if both client processes attempted to establish the connection simultaneously, then the connection could also become open. Both ends of the connection will transmit a system packet (with sequence number zero, and their Connection ID), causing the other end to become established. Delayed duplicates may cause the two ends to be established asymmetrically, preventing the connection from ever becoming open.

7.4.2 Consumer/server hookup

It will often be the case that a consumer client process wishes to communicate with a server client process that advertises a service on a known socket. The scenario for opening such a connection is as follows:

(1) The server establishes a *Service Listener* process at a well-known, or well-advertised socket. The Service Listener process accesses the Internet Transport Protocols package at the Internet Datagram Protocol (level one). The Service Listener process is willing to accept packets from any source.

(2) The consumer process creates an unestablished end of a connection using the appropriate interface (packet stream or byte stream). The consumer process attempts to open the connection, and therefore the protocol package transmits a packet to the well-known socket in order to establish the other end.

(3) Upon receipt of the consumer's packet, the Service Listener creates a new service supplying agent process, and creates one end of the connection that is unestablished. The new service supplying agent process will use an appropriate interface to the Sequenced Packet Protocol package. The Sequenced Packet Protocol package is then given the just arrived packet, whose Destination Socket is modified to reflect the socket being used by the new service supplying agent process. The Checksum will have to be modified appropriately as well. This packet will cause the end of the service supplying agent's connection to become established.

(4) The new service supplying agent process' protocol package returns a packet with the System Packet bit set, that causes the consumer's end to be established. Note that it is permissible for the consumer's end to be established to a socket number other than the

well-known socket used in the first packet. Communication may now proceed normally if the connection is open.

Throughout, the presence of the Connection ID makes it extremely unlikely that the consumer process' end of the connection will be falsely established, due to a delayed duplicate packet being delivered to the socket in the brief interval that the connection is unestablished. Even if it is falsely established, the connection will never become open because of the unique Connection ID picked by the consumer or protocol package for this incarnation of the connection.

Ensurance of correct operation at the server end is a little more complex: If the first packet is lost, no harm will result, since the sender will timeout and retransmit as with any other packet. If the first packet is received twice, the result depends on the implementation of the listener. A clever Service Listener will remember that it had recently seen a packet with the same Source Socket and Connection ID (by peeking into the level two field), and pass the duplicate packet to the recently started service supplying agent process or discard it. A simpler implementation will start up a new service supplying agent process and let one of them time out for lack of traffic. Even if the first packet from the consumer's process to the Service Listener contains data and many service supplying agent processes are created, they will not perform any irrecoverable action since the data will not be delivered unless it arrives in a packet that contains two Connection IDs or the data bearing packet is believable owing to subsequently received packets.

7.5 Terminating connections

The termination of a connection sometimes requires a little care. There are three separate but interlocking messages to transmit. First, there is the message signifying that all data has been sent. Second, there is the message signifying that all the data has been received and processed. Third, there is the message signifying that the sender understands and is turning to other things. Depending on whether the responsibility for ensuring that the transaction goes through resides with the sender or the receiver, some of these messages can often be omitted, but the general case requires all three. The above exchange cannot take place at the Sequenced Packet Protocol level, because it cannot know that the packets it has given to client software have been processed and saved in a "safe" place like a disk (see Appendix G).

A general way by which connections may be reliably terminated by two client processes involves a three way exchange of packets of Datastream Type "end" and "end-reply". The termination of a connection may be initiated by either client process by the transmission of a packet of type "end". The receiver of this packet responds with a packet of type "end-reply", and then proceeds to *dally* up to some reasonably long timeout (say 10 seconds). If and when the dallying end receives a packet of type "end-reply", it may destroy its end of the connection.

In the normal case, a Sequenced Packet Protocol packet of type "end" and two packets of type "end-reply" will be required to promptly close a connection. When the end that transmitted the packet of type "end" receives a packet of type "end-reply", it replies with a packet of type "end-reply". The purpose of the second packet of type "end-reply" is to inform the dallying end that it need dally no longer. Thus, in the normal case, termination requires three packets. In a slightly less normal case where the initiator's packet of type "end-reply" is unable to reach the dallying end before its resource-wasting timeout goes off, the initiator will conclude that the connection terminated properly while the dallying end will conclude that its packet at least

will be seen by the initiating process (the protocol package will tell the dallying end the last sequence number acknowledged by the initiating process' protocol package). In the even more unlikely case that the dallying end times out before the initiating process' protocol package receives the packet of type "end-reply", the dallying end will conclude that the connection terminated abnormally. The initiating process may conclude that the connection terminated properly if it receives and replies to the packet of type "end-reply" before its timeout goes off. Otherwise it, too, will conclude that the connection terminated abnormally.

It may happen that both client processes choose to terminate simultaneously. Upon receiving a packet with type "end" in seeming response to a packet of type "end" of its own, a client process should reply with a packet of type "end-reply" and begin dallying.

While such a protocol would be defined and implemented at level three in conjunction with the other control functions specific to that level three protocol, we have chosen to use the convention that clients should send packets of Datastream Type 254 (decimal) for "end" and 255 (decimal) for "end-reply". Such packets will have no data in them, but will consume sequence numbers. This convention makes it possible for all level three protocols to use a common set of level three library subroutines to implement the connection termination protocol (see Appendix G).

7.6 Incarnations of connections

The Sequenced Packet Protocol guarantees reliable communication once a connection is open. However, if packets can be delayed in the communication system for a time greater than the duration of a connection, and different incarnations of the same connection are used for communication in rapid succession, it is possible that packets from one incarnation will be confused with those from another. As a result, reliable communication between two client processes may not take place [16]. The use of connection IDs, however, makes it possible to distinguish one incarnation of a connection from another. The previous discussion on getting connections established and opened indicates some of the ways by which delayed duplicates can cause ends of a connection to be established asymmetrically, thereby prohibiting communication and the delivery of delayed duplicates. In the presence of a lot of delayed duplicates, two ends of a connection may thrash for a while before they become established symmetrically.

The state information and software complexity required to verify that packets are part of a new incarnation (as in the case of the intelligent Service Listener) can be reduced considerably if Connection IDs are always monotonically increasing. If Connection IDs are always monotonically increasing, then it is unnecessary for the intelligent listener to remember all Connection IDs seen for some interval of time, but rather only the last one seen from a particular source. The best way to guarantee that Connection IDs are monotonically increasing is to pick them from a clock register (that is unaffected by "booting" the machine) within the host. This guarantees that as long as the power is on, Connection IDs will be monotonically increasing. Hence, there is no dependence on the availability of non-volatile memory. The wrap-around time of the clock should be an order of magnitude greater than the maximum packet lifetime in the internet.

Mechanisms for establishing ends of a connection have been described, and packets will only be accepted if the connection is open (symmetrically established). It can, however, be proved that connections can be falsely opened (even with the Connection ID mechanism) causing erroneous acceptance of packets, in only two situations. The probability of these situations

arising is very small. The first situation is one in which both communicating client processes crash, and on restart choose the same connection ID as they had during the last incarnation. The second situation is one in which two client processes communicate for a long time, cycling through the sequence number space, terminate the connection, and then reestablish the connection choosing the same Connection ID. By having a clock with a fine grain, and relying on the stochastic nature of communication and the scheduling of the software processes within a host, the probability of these situations arising is very small, but finite, nonetheless.

7.7 Implementation notes

The Sequenced Packet Protocol has been designed to support both high and low bandwidth communication. The receiving end controls the flow with which data may be sent to it, and conversely the sending end controls the frequency with which the receiving end should return acknowledgements. Implementations should adhere to some well specified heuristics that control the request for acknowledgements, the generation of acknowledgements, and the retransmission of packets. These functions are interrelated. We specify some important guidelines. Even though data may flow in both directions on the connection, the following discussion will describe the flow of data from a *sending end* to a *receiving end*, with the assumption that the same comments hold for data flowing in the opposite direction.

The sending end may request an acknowledgement at any time. Typically, it sets the Send Acknowledgement bit in a packet just before transmitting the data packet with the sequence number corresponding to that permitted by the receiving end's allocation. The sending end may transmit a *probe*, which is a system packet with the Send Acknowledgement bit set, to ask the receiving end for an allocation and/or an acknowledgement.

If there is data flowing in the opposite direction, the sending end will automatically receive acknowledgements and allocations. Otherwise, the receiving end should respond with a system packet (that contains an acknowledgement) whenever the sender asks for one. The receiving end should not turn around and do so immediately, but rather should remember that it was asked for an acknowledgement, and then process packets that have queued up before returning an acknowledgement and an allocation.

The receiving end should not generate system packets gratuitously. The Sequenced Packet Protocol has been designed to let the sender of data control the return of acknowledgements (in a situation with one way flow of data), and while it is permissible for the receiving end to occassionally generate gratuitous system packets when some exceptional condition occurs (it had not supplied any allocation and now can supply a large number), this should not be done as a rule. The reason behind this recommendation is that the sending end has no effective way to determine whether the system packet containing an acknowledgement that it receives is in response to a request, a gratuitous response, or a delayed packet. Ill-timed gratuitous system packets may cause the sending end to open its flow control window by small ammounts, thereby forcing it to ask for an acknowledgement every so often, and reducing the effective data throughput. If the sending end has had all its packets acknowledged, but has not received an allocation for a long time, it can send a data packet or a probe in an attempt to discover whether the receiving end has increased its allocation.

The sending end should have an estimate of the roundtrip delay on the connection and retransmit unacknowledged packets appropriately. Roundtrip delays may be determined by setting the Send Acknowledgement bit and then waiting for the response. (This roundtrip delay may be multiplied by 2 to determine the time interval before retransmitting

unacknowledged packets.) This technique does not always result in the right estimate for two reasons: (1) the sending end is not sure whether the response was generated gratuitously or not, and (2) the delay at the receiving end changes with time. Packets that are retransmitted should not all have their Send Acknowledgement bit set, since this may cause excessive transmissions of system packets from the receiving end. In general, either the last packet retransmitted should have its Send Acknowledgement bit set, or the same packets that had the bit set the first time they were transmitted.

An implementation of the Sequenced Packet Protocol should generate and process Error Protocol packets to enhance the service provided to client software. For example, a Sevice Listener (see Section 7.4.2) should generate an Error Protocol packet when it receives a connection request that it can not satisfy.

8

Level two: Packet exchange protocol

8.1 Introduction

The *Packet Exchange Protocol* is used to transmit a request in a packet and receive a response with reliability *greater* than that achievable by transmitting internet packets directly as datagrams, and *less* than that achievable through use of the Sequenced Packet Protocol. This simple protocol is "single packet" oriented, and assumes that the client is capable of dealing with this level of unreliablity. This protocol retransmits a request, but does not perform any duplicate detection; it is suitable in applications where the request/response transactions are idempotent, or where the communicating clients are capable of providing the necessary level of reliablity through the very nature of their application and form of interaction.

Packet Exchange Protocol packets may be sourced at and destined to any socket. As in the case of the Sequenced Packet Protocol, the use of a well-known socket is higher level protocol-dependent and must be registered. The format of a Packet Exchange Protocol packet is illustrated in Figure 8.1 and described in detail below. The packet type of such internet packets is *packet exchange protocol*, which is specified in Appendix E.

8.2 Packet format

ID

The *ID* is a 32-bit field, in which the "requestor" can place a "transaction identifier" and cross check it with the identifier that the "replier" returns. If they are the same, then the request/response pair of packets completes the transaction. This field should be viewed as a monotonically increasing number in which the most significant bit is Bit 0 it of the first word. The wrap around time of this field should be an order of magnitude greater than the maximum packet lifetime in the internet.

Client Type

The *Client Type* contains a registered value that identifies the client of the protocol. This is the bridge between this level and higher level protocols built using it. The procedure for registering client types for the Packet Exchange Protocol is specified in Appendix F.

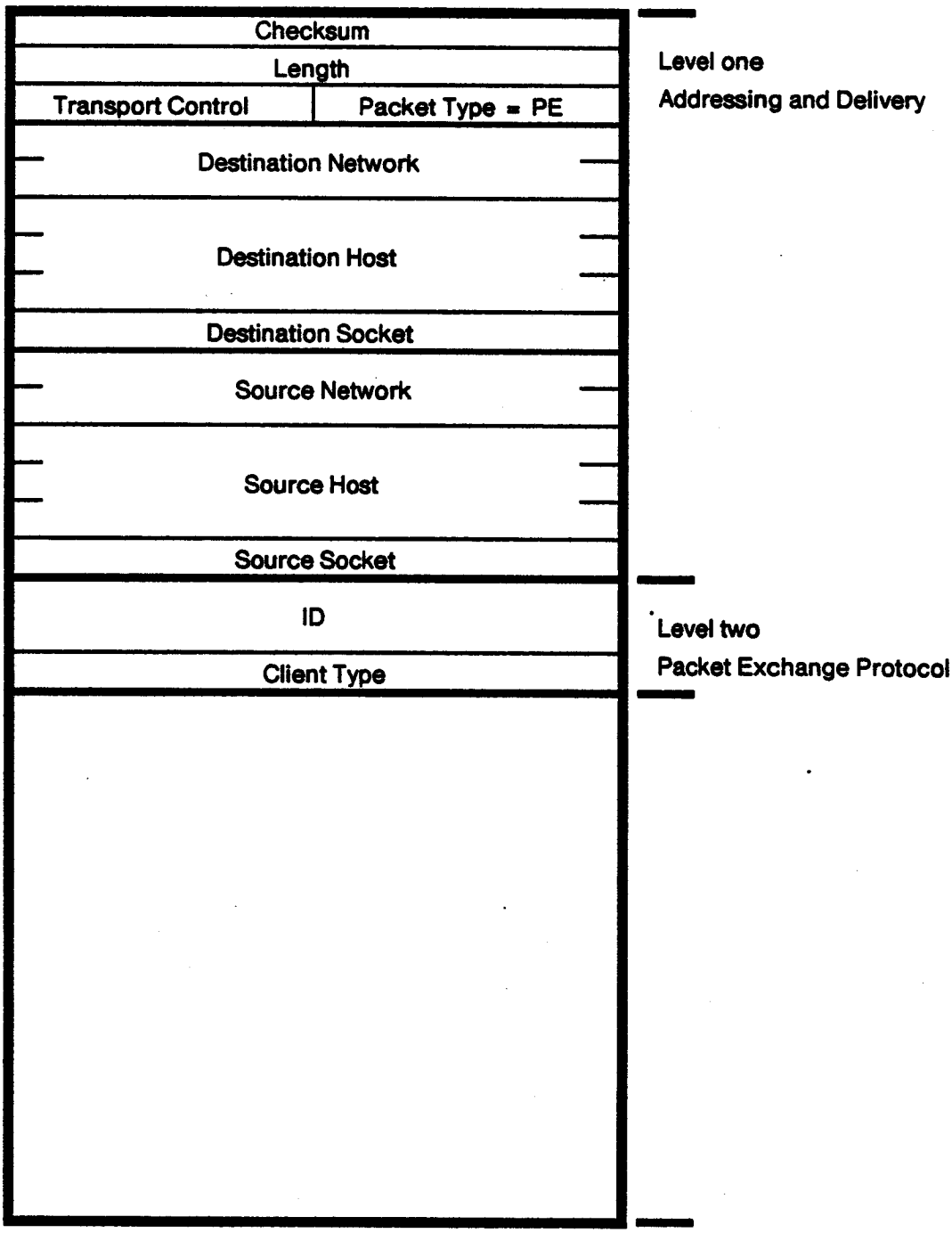


Figure 8.1 The packet exchange protocol packet

8.3 Implementation notes

The requesting client of this protocol specifies that the contents of such a packet be delivered to the specified destination network address. This packet contains a registered Client Type, is assigned an ID, and is transmitted to the specified address. The client now waits at its local socket for a Packet Exchange Protocol packet with the same ID. If such a packet arrives, the client can verify its appropriateness by the contents of the Client Type field. For example, if the requestor wants the time of day, it will send such a packet to a time of day server with "time of day request" in the Client Type field, and will expect a reply with "time of day" in the Client Type field and the time of day and other relevant information in the data field. The implementation of this protocol retransmits the request packet for a client-specified time. The retransmission interval is chosen by the protocol package based on delay estimates from the router to the remote network address, and a client-specified estimate of response time at the responding host.

At the replying end, client software waits for Packet Exchange Protocol packets and performs the requested operation based on the client type field. The replier may get duplicate requests, and must reply, since it has no way of knowing that a previous reply made it back to the requestor.

Classes of requests may be grouped together at one or more (well) known sockets. Client Type must be unique throughout all of the Network Systems so there will be no confusion in the operation caused by a request. Of course, clients of this protocol are free to use an unregistered client type at a socket that is private to them, for purposes of experimentation and development.

A

Appendix A
References

[1] Abraham, S. M.; Dalal, Y. K. Techniques for Decentralized Management of Distributed Systems. *20th IEEE Computer Society International Conference (Compcon).* 430-436; 1980 February.

[2] Boggs, D. R.; Shoch, J. F.; Taft, E. A.; Metcalfe, R. M. PUP: An internetwork architecture. *IEEE Transactions on Communications.* Com-28(4): 612-624; 1980 April.

[3] Boggs, D. R. Internet Broadcasting. Palo Alto: Stanford University; 1981. Ph.D. Thesis in preparation (will be available from Xerox Corporation, Palo Alto Research Center).

[4] Cerf, V. G.; Kirstein, P. K. Issues in Packet-Network Interconnection. *Proceedings of the IEEE.* 66(11): 1386-1408; 1978 November.

[5] Cohen, D. On Holy Wars and a Plea for Peace. To be published in *IEEE Computer* magazine; 1981.

[6] Consultative Committee, International Telephone and Telegraph [C.C.I.T.T.]. *Recommendation X.25, Interface Between Data Terminal Equipment (DTE) and Data Circuit Terminating Equipment (DCE) for Terminals Operating in the Packet Mode on Public Data Networks.* Geneva: International Telecommunications Union. v. 7 [orange book].

[7] Dalal, Y. K.; Printis, R. S. 48-bit Absolute Internet and Ethernet Host Numbers. Palo Alto: Xerox Corporation, Office Products Division; 1981 July; OPD-T8101. To be published in the *Proceedings of the Seventh Data Communications Symposium.* 1981 October.

[8] Digital Equipment Corporation; Intel Corporation; Xerox Corporation. The Ethernet, A Local Area Network: Data Link Layer and Physical Layer Specifications. 1980 September 30; Version 1.0.

[9] International Organization for Standardization. *ISO Open Systems Interconnection—Basic Reference Model.* ISO/TC 97/SC 16 N 719; 1981 August.

Marked-up Reference Model of OSI (DP 7498) to Add Connectionless Data Transmission. ISO/TC 97/SC 16 N 741; 1981 September.

[10] Oppen, D. C.; Dalal, Y. K. The Clearinghouse: A Decentralized Agent for Locating Named Objects in a Distributed Environment. Palo Alto: Xerox Corporation, Office Products Division; 1981 October; OPD-T8103.

[11] Postel, J., ed. *DoD Standard Internet Protocol. ACM Computer Communication Review.* 10(4): 2-51; 1980 October. Available from NTIS, Springfield, Virginia; ADA079730.

[12] Postel, J., ed. *DoD Standard Transmission Control Protocol. ACM Computer Communication Review.* 10(4): 52-132; 1980 October. Available from NTIS, Springfield, Virginia; ADA082609.

[13] Redell, D. D.; Dalal, Y. K.; Horsely, T. R.; Lauer, H. C.; Lynch, W. C.; McJones, P. J.; Murray, H. G.; Purcell, S. C. Pilot: An Operating System for a Personal Computer. *Communications of the ACM.* 23(2): 81-92; 1980 February.

[14] Shoch, J. F. Packet Fragmentation in Internetwork Protocols. *Computer Networks.* 3(1); 1979 February.

[15] Shoch, J. F.; Cohen, D.; Taft, E. A. Mutual Encapsulation of Internetwork Protocols. *Proceedings, Trends and Applications: 1980—Computer Network Protocols.* 1-11; 1980 May. Revised version to appear in *Computer Networks.*

[16] Sunshine, C. A.; Dalal, Y. K. Connection Management in Transport Protocols. *Computer Networks.* 2(6): 454-473; 1978 December.

[17] Zimmermann, H. OSI Reference Model—The ISO Model of Architecture for Open Systems Interconnection. *IEEE Transactions on Communications.* Com-28(4): 425-432; 1980 April.

B

Appendix B
Host number assignment procedures

A *host number* is 48 bits long and can either be a physical host number or a logical host number, that is, a multicast ID. All system elements have unique physical host numbers. If a system element is connected to one or more Ethernets, then the Ethernet address on each of the Ethernets is the same as its physical host number. Since most system elements will be connected to Ethernets, the acquisition procedures for 48-bit host numbers are identical to the acquisition procedures for 48-bit Ethernet addresses, as described in *The Ethernet, A Local Area Network: Data Link Layer and Physical Link Layer Specifications* [8]. Host numbers are assigned in blocks of 33,554,432 ($2*2^{24}$)—half are physical and half are logical. All the physical host numbers within a block must be used, that is, host numbers should be used densely. To obtain a block of host numbers contact:

Xerox Corporation
Office Products Division
Network Systems Administration Office
3333 Coyote Hill Road
Palo Alto, California 94304

As described in [8], the 48-bit Ethernet address is represented as a sequence of six 8-bit bytes: A, B, C, D, E, F. The bytes are transmitted on the Ethernet in the order A, B, C, D, E, F with the least significant bit of each byte transmitted first. The least significant bit of byte A is the multicast bit, identifying whether the 48-bit number is a physical or logical host number. Figure B.1 illustrates how the bytes of a 48-bit host number are layed out in an internet packet.

Note that all level one (and higher) packet formats in this document are drawn as they appear in the memory of a processor that has a 16-bit word, and where the first word of the packet is in memory location *m*, and the next word in location *m+1*. The bits (or bytes) within a word are numbered left to right. The process of encapsulation and decapsulation *must* preserve this logical structure. An internet packet is converted into an Ethernet packet (a sequence of 8-bit bytes) by going left to right within a word, and going from word *m* to word *m+1* within the packet. Processors with different memory addressing techniques should convert an internet packet into an Ethernet packet appropriately, thereby preserving the order of bits on the wire. Problems associated with transforming a packet in memory to a bit-stream are discussed in [5].

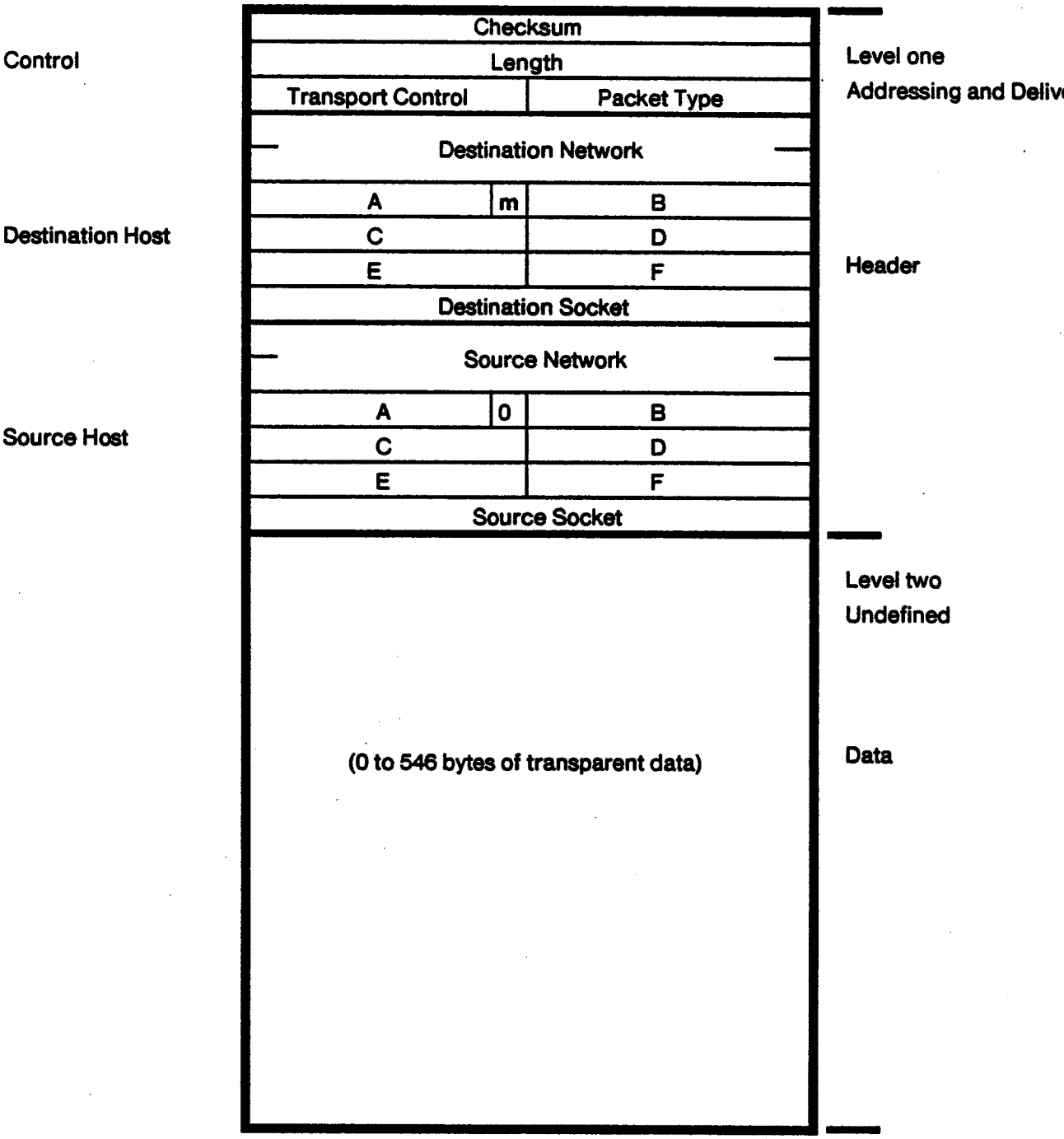


Figure B.1 Host address layout in an internet packet

C

Appendix C
Network number assignment procedures

The network number space is flat. The network numbers *zero* and *all ones* are reserved to mean *unknown* and *all*, respectively.

To obtain a block of network numbers contact:

Xerox Corporation
Office Products Division
Network Systems Administration Office
3333 Coyote Hill Road
Palo Alto, California 94304

Appendix D
Assigned well-known socket numbers

The socket numbers *zero* and *all ones* are reserved to mean *unknown* and *all*, respectively. Well-known socket numbers have the range 1 to 3000 decimal. All other socket numbers are ephemeral, that is, they may be dynamically assigned and reused. Well-known sockets that are used by the protocols in this specification are:

Function	Well-Known Socket (octal)
Routing Information	1
Echo	2
Router Error	3
Experimental	40-77

Experimental well-known socket numbers may be used when developing new applications.

To obtain a well-known socket number, contact:

Xerox Corporation
Office Products Division
Network Systems Administration Office
3333 Coyote Hill Road
Palo Alto, California 94304

E

Appendix E
Assigned internet packet types

The internet packet type *zero* means *unknown.* The internet packet types used by protocols in this specification are:

Protocol	Packet Type (octal)
Routing Information	1
Echo	2
Error	3
Packet Exchange	4
Sequenced Packet	5
Experimental	20-37

Experimental internet packet types may be used when developing new applications.

To obtain an internet packet type, contact:

Xerox Corporation
Office Products Division
Network Systems Administration Office
3333 Coyote Hill Road
Palo Alto, California 94304

F

Appendix F
Packet exchange client type assignment

Experimental Packet Exchange client types, numbered 40 through 77, may be used when developing new applications.

To obtain a Packet Exchange client type, contact:

Xerox Corporation
Office Products Division
Network Systems Administration Office
3333 Coyote Hill Road
Palo Alto, California 94304

G

Appendix G
Level three: Connection termination protocols

In the past, most connection-oriented protocols have included a mechanism to terminate or close the connection [16; 2]. The Sequenced Packet Protocol, however, does not include such a mechanism and relies on its clients to determine that they have finished (see Section 7.5).

This decision was based on the fact that the semantics of termination is application-specific. If the connection protocol imposes a semantics, then the protocol is not general, and at many times of little direct use to the clients. In addition, the protocol machines at each end become very complex [16]. Since connection termination initiated by either client of a full-duplex connection requires the active participation of both clients, the Sequenced Packet Protocol pushes the responsibility entirely to its clients; thereby simplifying both the Sequenced Packet Protocol and the interaction between the protocol package and its client.

While there may be many ways for the communicating clients to terminate the connection, we recommend that in the most typical case they use a simple level three protocol that makes use of Datastream Types, and is described in Section 7.5.

XEROX

Xerox Corporation
Stamford, Connecticut 06904