Pup: An Internetwork Architecture

DAVID R. BOGGS, MEMBER, IEEE, JOHN F. SHOCH, MEMBER, IEEE, EDWARD A. TAFT, AND ROBERT M. METCALFE

(Invited Paper)

Abstract—**Pup is the name of an internet packet format (*PARC Universal Packet*), a hierarchy of protocols, and a style of internetwork communication. The fundamental abstraction is an end-to-end media-independent internetwork datagram. Higher levels of functionality are achieved by end-to-end protocols that are strictly a matter of agreement among the communicating end processes.**

This report explores important design issues, sets forth principles that have guided the Pup design, discusses the present implementation in moderate detail, and summarizes experience with an operational internetwork. This work serves as the basis for a functioning internetwork system that provides service to about 1000 computers, on 25 networks of 5 different types, using 20 internetwork gateways.

I. INTRODUCTION

RESEARCH in network interconnection techniques has been motivated by the desire to permit communication among diverse, geographically distributed computing resources and users interconnected by a wide variety of network technologies.

It is the purpose of an internetwork architecture to provide a uniform framework for communication within a heterogeneous computing, communication, and applications environment. The work described in this paper represents one internetwork architecture, known as *Pup*, in widespread regular use within Xerox. The name referred originally to the abstract design of a standard internetwork datagram (the PARC Universal Packet), but has expanded in usage to include the whole hierarchy of internetwork protocols as well as a general style for internetwork communication.

To assist in understanding the design of the Pup protocols, it is useful to characterize briefly the environment in which this architecture has evolved.

The computational environment includes a large number of "Alto" minicomputers [11], [31], and other personal computers capable of high-quality interaction with human users. Supporting these are various specialized server systems that are shared among many users and provide access to expensive peripherals such as large disks, magnetic tapes, and high-quality printers. Additionally, there are several general-purpose time sharing systems providing customary services for terminal users.

The communications environment includes several different individual network designs. The dominant one is the "Ethernet" communications network, a local-area broadcast channel with a bandwidth of 3 Mbits/s [15]. Long-haul communication facilities include the ARPANET, the ARPA packet radio network, and a collection of leased lines implementing an ARPANET-style store-and-forward network. These facilities have distinct native protocols and exhibit as much as three orders of magnitude difference in bandwidth.

The applications to be supported include a wide range of activities: terminal access to the time sharing services, electronic mail, file transfer, access to specialized data bases, document transmission, software distribution, and packet voice, to name just a few. We would also like to facilitate more ambitious explorations into the area generally referred to as "distributed computing."

This paper is organized as follows. In Section II we discuss some of the design issues which have emerged in the formulation of the Pup architecture, while Section III provides more detail on the protocols themselves. Section IV describes briefly some of our operational experience with the present implementation. The final section presents a retrospective critique of the work, highlighting some areas which merit further attention.

II. DESIGN PRINCIPLES AND ISSUES

Constructing an architecture for internetwork protocols is, first and foremost, an exercise in design: identifying individual issues, exploring alternative solutions, and then knitting these pieces together to form the final result. Along the way, many compromises are made as one trades off among different criteria: functionality, efficiency, generality, ease of implementation, extensibility, and others.

In this section we enumerate some of the major design issues confronted in the development of a network architecture and describe, in general terms, the choices made in the development of Pup. (Several of these and other issues are enumerated in [2] and [17]). From this discussion the broad outlines of Pup will emerge; the section that follows provides more specific detail about the actual design.

A. The Basic Model: Individual Networks Connected with Gateways

As with most internetwork models, one envisions a collection of heterogeneous networks, connected with a set of *internetwork gateways* to form a loosely coupled system known generally as an *internet* [1], [2], [26]. An internet should provide the ability for any two hosts to communicate, so long as their own local networks are interconnected.

Manuscript received May 29, 1979; revised November 2, 1979.

D. R. Boggs, J. F. Shoch, and E. A. Taft are with the Palo Alto Research Center, Xerox Corporation, Palo Alto, CA 94304.

R. M. Metcalfe was with the Palo Alto Research Center, Xerox, Corporation, Palo Alto, CA 94304. He is now with the Computer Systems Laboratory, Stanford University, Stanford, CA 94305 and 3Com Corporation, 3000 Sand Hill Road #1, Menlo Park, CA 94025.

0090-6778/80/0400-0612$00.75 © 1980 IEEE

An important feature of the Pup internet model is that the hosts *are* the internet. Most hosts connect directly to a local network, rather than connecting to a network switch such as an IMP, so subtracting all the hosts would leave little more than wire. Gateways are simply hosts in the internet that are willing to forward packets among constituent networks. Thus, most of the properties of the internet are primarily artifacts of host software. The architecture must scale gracefully, and in particular must allow for the existence of a degenerate internet consisting of a single local network and no gateways.

B. Simplicity

One of the guiding principles in designing Pup has been the desire for simplicity. Pup is a framework for computer communications research, and simplicity is one of the best ways to minimize restrictions and maximize flexibility for experimentation. Attempting deliberately to eliminate unneeded complexity helps to keep the design open-ended. This in turn makes it easier to incorporate the existing diverse collection of networks and hosts and to accommodate new alternatives as the technology matures. Keeping the design simple helps to avoid building in technological anachronisms.

A second motivation for this principle is the desire to foster efficient implementations of the protocols in the host machines, which are typically quite small. Software overhead must be kept low in order to sustain high-bandwidth local communication, which constitutes the bulk of the traffic; yet the same software must support the full generality of internetwork communication.

C. Datagrams versus Virtual Circuits

There are two major approaches to providing an interface to packet-switched communications: accepting individual *datagrams* or providing a higher level of service in the form of a *virtual circuit.* The two interfaces are not unrelated, since a virtual circuit interface is usually implemented within a network by the use of datagrams. In some sense, datagrams provide access to a network at a lower level, closer to its underlying capabilities. Datagrams are particularly useful in many kinds of transaction-oriented protocols. Furthermore, the task of the internet is significantly simplified if it need only transport independent, individually-addressed datagrams, without having to maintain the state required to support virtual circuits. If the internet provides a datagram interface, virtual circuit interfaces can be provided by adding appropriate mechanisms at the end points.

Therefore, the basic function provided by the Pup internet is the transport of datagrams; this simple abstraction is the foundation of Pup. The internet does not guarantee reliable delivery of datagrams (called "Pups"); it simply gives its "best efforts" to deliver each one, and allows the end processes to build protocols which provide reliable communications of the quality they themselves desire [14]. The internet has no notion of a connection. It transports each Pup independently, and leaves construction of a connection—if that is the appropriate interprocess communication model—to the end processes. Keeping fragile end-to-end state out of the packet transport system contributes to reliability and simplicity.

D. Individual Networks as Packet Transport Mechanisms

Individual networks within the internet can be viewed simply as *packet transport mechanisms.* As links in the internet they give their best efforts to deliver internet packets, but they do not guarantee reliable delivery. Packets may be lost, duplicated, delivered out of order, after a great delay, and with hidden damage. A network can have any combination of bandwidth, delay, error characteristics, topology, and economics; the routing algorithm should attempt to take these characteristics into consideration.

Encapsulation is an invertible, network-dependent transformation performed on a Pup to permit it to be carried transparently through a network: an abstract Pup is presented at one end, encapsulated for transmission through the net, and decapsulated at the other end, yielding the original Pup. For some networks, encapsulation consists merely of adding headers and trailers. More elaborate transformations may be necessary to pass a Pup through other networks (for example, using low-level acknowledgments or error correction because the network has a high loss rate). Encapsulation and decapsulation take place in a *network-specific driver* in which is vested all knowledge of the encapsulation technique. The internet specification has nothing to say about encapsulation except that it be invisible.

E. Internetwork Gateways

We distinguish two kinds of gateways: *media translators* and *protocol translators.* Media gateways are hosts with interfaces to two or more packet transport mechanisms among which they forward internet datagrams, using the appropriate encapsulation for each. These are the heart of any datagram-based internet. Protocol gateways are hosts which speak two or more functionally similar but incompatible higher-level protocols used to transport information within networks, mapping one higher level abstraction into the other. (It is clear that a media gateway is just doing protocol translation at the link level, but the distinction is useful given the importance of internet datagrams in this architecture.)

In the Pup internet, media gateways are by definition simple, since all that is required of the translation process is that it preserve the semantics of internetwork datagrams. Protocol gateways are usually more difficult, even when the protocols are similar, since such higher level protocols provide richer and more specialized semantics and it is not always clear how one should map the functionality of one protocol into another. Development of higher level protocol translators between different network and internet architectures, e.g., between the ARPANET file transfer protocol (FTP) and the Pup-based FTP, is a thorny task, but one that must be confronted when interconnecting systems that do not share the necessary lower level primitives.

F. A Layered Hierarchy of Protocols

Layering of protocols is one of the most effective means for structuring a network design: each level uses the functions of the lower level, and adds some functionality of its own for possible use by the next level. Provided that suitable interfaces are maintained, an implementation at one level can be modified without impacting the overall structure; this helps to simplify both the design and the implementation.

Pup protocols are organized in a hierarchy, as shown in Fig. 1; the details of this figure will be presented in Section III. A level represents an abstraction, to be realized in different ways in different hosts. There are four levels of interest, but there may be more than one protocol at any level except level 1, representing a different use of the underlying layers. (The numbering of layers—and, indeed, the choice of points at which to divide the layers—is arbitrary; there is no relationship between Pup's numbering and that of other designs such as the Open Systems Architecture.)

The level 0 abstraction is a packet transport mechanism. There are many realizations: an Ethernet channel, the ARPANET, the ARPA packet radio network, our store-and-forward leased line network, and others. Level 0 protocols include specifications such as hardware interfaces, electrical and timing characteristics, bit encodings, line control procedures, and network-dependent packet formatting conventions. Associated with each packet transport mechanism is a convection for encapsulating Pups.

The level 1 abstraction is an internet datagram. The realization of this abstraction consists of the format of a Pup, a hierarchical addressing scheme, and an internetwork routing algorithm. There is only one box at level 1: the internet datagram protocol; it is this layer of commonality which unifies all of the different networks that might be used at level 0, and which makes available a uniform interface to all of the layers above. It is the purpose of this level to provide media independence while maintaining the common properties of the underlying packet networks.

The level 2 abstraction is an interprocess communication mechanism: a way to move bits without saying much about their form or content. Various level 2 protocols provide many combinations of reliability, throughput, delay, and complexity. These protocols can be divided into two classes according to the amount and lifetime of state information kept by the communicating end processes. Connectionless protocols support short-lived interactions; the end processes maintain little state, and usually only during the exchange of a few Pups—no more than a few seconds. Connection-based protocols support sustained interactions, generally requiring substantial state to be maintained at both ends, and for longer periods—minutes to hours.

Level 3 adds structure to the data moved at level 2, as well as conventions for how processes interact. For example, the file transfer protocol (FTP) consists of a set of conventions for talking about files and a format for sending them through a level 2 byte stream protocol connection. These are sometimes referred to as function-oriented protocols [4].

Above level 3 the dividing lines become blurred, and individual applications evolve with their own natural decomposition into additional layers. With respect to layering of protocols, Pup is similar in many ways to the ARPA internet and TCP design [1] and the Open Systems Architecture [32]. Unlike the Open Systems Architecture (and others), Pup often has several alternative boxes which all rest on a lower level and offer different functionality and interfaces to the next higher level.

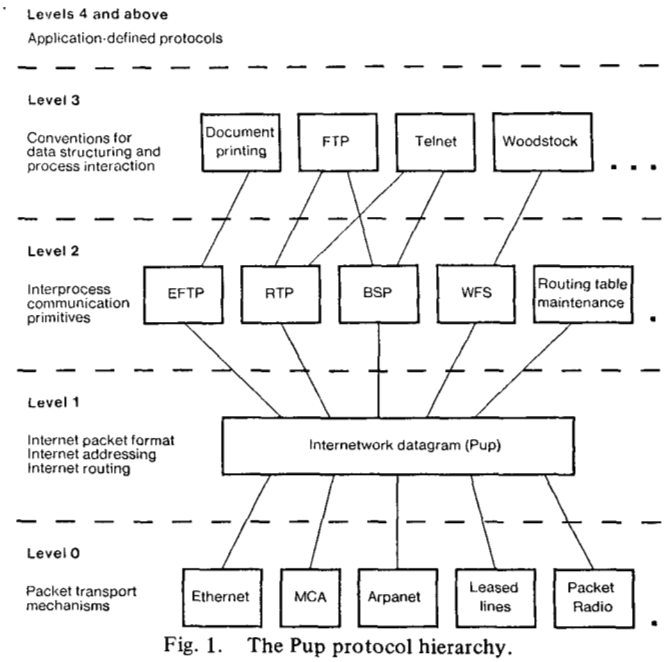

Fig. 1. The Pup protocol hierarchy.

G. Naming, Addressing, and Routing

Names, addresses, and routes are three important and distinct entities in an internet [19]:

> The *name* of a resource is *what* one seeks,
> an *address* indicates *where* it is, and
> a *route* is *how to get there.*

A name is a symbol, such as a human-readable text string, identifying some resource (process, device, service, etc.). An address is a data structure whose format is understood by level 1 of the internet, and which is used to specify the destination of a Pup. A route is the information needed to forward a Pup to its specified address. Each of these represents a tighter binding of information: names are mapped into addresses, and addresses are mapped into routes. Error recovery should successively fall back to find an alternate route, then an alternate address, and then an alternate name.

The mapping from names to addresses is necessarily application-specific, since the syntax and semantics of names depend entirely on what types of entities are being named and what use is being made of them. This is dealt with at the appropriate higher levels of protocol.

An address field, as contained in a Pup, is one of the important elements of commonality in the internet design. An end process sends and receives Pups through a *port* identified by a hierarchical address consisting of three parts: a *network number*, a *host number*, and a *socket number*. This structure reflects the attitude that the communicating parties are the

end processes, not the hosts' protocol handlers; among other things, this permits alternate implementations of a higher level protocol to coexist in a single machine. (In contrast, the ARPA internet Project [17] takes the position that the socket abstraction does not belong at the internet level; therefore, ARPA Internet addresses contain only network and host numbers. When a packet arrives, it is first demultiplexed by the *protocol type* field in the internet header; higher level protocols such as the TCP, datagram protocol, and packet voice protocol then impose their own concept of socket if they find it useful, which, as a practical matter, they all do.)

The actual process of routing a packet through the Pup internet uses a distributed adaptive routing procedure. The source process specifies only the *destination address* and not the *path* from source to destination. The internetwork gateways route Pups to the proper network, a network then routes Pups to the proper host, and a host routes Pups to the proper socket.

This routing process is associated with level 1 in the protocol hierarchy, the level at which packet formats and internet addresses are standardized. The software implementing level 1 is sometimes referred to as a *router*. Thus, the routing table itself is kept at level 1; a very simple host (or gateway) would need only levels 0 and 1 in order to route Pups. But the routing table also requires periodic updating, as gateways exchange and distribute their current routing information; this *routing table maintenance* protocol is found logically at level 2 of the hierarchy.

Gateways provide internet routing tables to individual hosts as well as to each other. Hosts use this routing information to decide where to send outgoing packets destined other than to a directly-connected network.

H. Flow Control and Congestion Control

Although the terms are often confused, *flow control* and *congestion control* attack two very different problems in packet-switched communication. Flow control is a mechanism used to regulate the behavior of a specific source and destination pair, so that the source does not send data at a rate greater than the receiver can process it. In an internet architecture, flow control remains the responsibility of the end-to-end protocols, particularly those at level 2 supporting regular stream traffic.

Congestion control is a network-wide mechanism, used to control the number and distribution of packets in the network so as to prevent system overload. Internet congestion control is necessary to help protect the gateways from being burdened with excessive traffic.

The Pup datagram-based internet model does not require that the internet successfully deliver every packet that has been accepted. Therefore, an intermediate gateway which suddenly encounters a period of severe congestion is free to discard packets, although the system should be engineered to make this an uncommon event.

If a gateway is forced to discard an incoming packet because of congestion, it should attempt to return some informato the source: an *error Pup* (negative acknowledgment) indicating that a packet had to be discarded in midroute. This error Pup is simply returned to the source port, as identified in the discarded Pup; this is a good illustration of the value of including the socket number as part of the standard internet address. The source process can use this information to modify its transmission strategies, for example, to reduce its offered load (the rate at which it attempts to send Pups along the congested path) and thereby help to relieve the congestion.

Long-term congestion should eventually be reflected in the routing information exchanged among gateways, discouraging subsequent traffic from attempting to pass through a badly congested area.

I. Reliable Transport

Defining datagrams to be less than perfectly reliable is realistic since it reflects the characteristics of many existing packet transport mechanisms. Probabilistic transmission is basic to the theory of operation of network designs such as Ethernet. Even in networks nominally designed to deliver correctly sequenced, error-free packets, occasional anomalies may result from certain hardware or software failures: an ARPANET IMP may crash while loading the only copy of a packet, or an X.25 virtual circuit may be reset.

As mentioned previously, the Pup internet *always* has the option of discarding packets to relieve congestion, although this is certainly not an optimal strategy. This point is of considerable practical importance when one considers the complicated measures required to avoid deadlock conditions in the ARPANET, conditions which are a direct consequence of attempting to provide reliable delivery of every packet in a store-and-forward network [13], [14]. Packet management strategies that attempt to guarantee perfect reliability must be designed to operate correctly under *worst case* conditions, whereas strategies that have the option of discarding packets when necessary need operate correctly only under *most* conditions. The idea is to sacrifice the guarantee of reliable delivery of individual packets and to capitalize on the resulting simplicity to produce higher reliability and performance overall.

For some applications, perfectly reliable transport is unnecessary and possibly even undesirable, especially if it is obtained at the cost of increased delay. For example, in real-time speech applications, loss of an occasional packet is of little consequence, but even short delays (or worse, highly variable ones) can cause significant degradation [3], [24].

Reliable delivery requires maintaining state information at the source and destination. The actions of a large class of simple servers, such as giving out routing tables or converting names into addresses, are idempotent (i.e., may be repeated without incremental effects), and a client of that service can simply retransmit a request if no response arrives. These protocols reduce to a simple exchange of Pups, with an occasional retransmission by the client, but with no state retained by the server. (The server may choose to retain answers to the last few requests to improve response time, but this optimization is invisible to the protocol.)

On the other hand, many applications such as file transfer and terminal connection do depend upon fully reliable transmission. In these cases, it is perfectly reasonable to build a

reliable end-to-end protocol on top of the internet datagrams. Ultimately, reliability (by some definition) is always required; the issue is where it should be provided. The Pup attitude is that it is the responsibility of the end processes to define and implement whatever form of reliable transport is appropriate to the situation.

J. Packet Fragmentation

It is inevitable that some process will want to send an internet packet which is too large to be directly encapsulated for transmission through an intermediate network that has a smaller maximum packet size. This problem is usually approached with one of two forms of *packet fragmentation* [20].

With *internetwork fragmentation*, an internet-wide design specifies the operations to be performed on a packet that is too large for a network it is about to enter. The internet datagram is fragmented into a number of smaller internet datagrams, thereafter to be transported independently and reassembled at the ultimate destination. This is the approach taken, for example, in the ARPA internet design. It requires every destination to have procedures for reassembly.

Alternatively, one may use *intranetwork fragmentation* (or *network-specific fragmentation*): when presented with an oversize packet, the network-specific driver undertakes to fragment the packet in a manner specific to that network, to be reassembled by the corresponding driver as the packet exits the network (e.g., at the next gateway). This approach confines the fragmentation and reassembly procedures to the level 0 modules of hosts directly connected to the network in which fragmentation is required.

The Pup design does not attempt to provide any form of general internetwork fragmentation. This complex issue has been simply legislated out of existence by requiring that every agent in the internet handle Pups up to a standard maximum size, using network-specific fragmentation where necessary.

K. Broadcast Packets

Broadcast packets are a particularly useful means for locating available resources or distributing information to many hosts at once. Some local networks, such as the Ethernet, directly support transmission of broadcast packets. In store-and-forward systems, however, specialized algorithms are required to propagate a packet efficiently to all hosts [5], [6]; no existing store-and-forward networks support any technique besides brute-force transmission of a packet to every node, although such a capability is now being implemented in the ARPANET.

Broadcasts may also be expensive since every host that receives one must expend some resources, if only to discard it. In networks where a broadcast involves generating more than one packet, there is the additional cost of creating and transporting the extra copies. Because of their potentially high cost, internet-wide broadcasts are not presently supported in the Pup design. Nor is it clear that such a capability would be desirable, since it would not extend well to a very large internet. The problem of locating distant resources in the internet at reasonable cost is a topic of current research.

But Pups can be broadcast on a single network; they are frequently used to locate nearby resources, or to permit gateways to announce their presence on a network. Implementation of the broadcast procedure is left to the network-specific driver, using the best technique available on that net.

L. Privacy and Security

It must be recognized that in practical internet environments, packets may be delivered to the wrong host, intercepted by another host, or generated by a host masquerading as some other host. To prevent this would require one to interpose some agent between hosts and the internet and to specify a secure access control procedure. This would significantly increase the complexity of the internet, and truly suspicious users would probably not trust it anyway.

Processes are encouraged, however, to ensure the privacy and authenticity of their communication by whatever end-to-end encryption techniques seem appropriate [16]. Particularly vulnerable components, such as gateways and servers, should take precautions to protect their own integrity, but ultimate responsibility rests with the end processes. The Pup internet does not attempt to protect users from traffic analysis or from malicious replay of previous traffic.

III. IMPLEMENTATION

The preceding section has outlined some of the important properties of the Pup architecture and the internetworking issues it addresses. What follows is a more detailed description of the present design of the four major layers in the system.

A. Level 0: Packet Transport

An individual network moves network-specific packets among hosts; the addressing schemes, error characteristics, maximum packet sizes, and other attributes of networks vary greatly. An internetwork packet transport mechanism, however, moves Pups between hosts. The level 0 code which transforms a network into an internet packet transport mechanism is called a *network driver*.

A machine connected to a single network, therefore, has one level 0 network driver; a gateway has one driver for each directly-connected network. Only the driver knows about the peculiarities of a network's hardware interface and low-level protocol.

The interface between levels 0 and 1 is very simple. Level 1 passes down a Pup and a network-specific host address, and the driver encapsulates the Pup and does its best to deliver it to the specified host. When a Pup arrives at a host, the driver decapsulates it and passes it up to level 1; if for any reason the Pup looks suspicious (as determined by network-specific error checking), the driver discards it.

Every packet transport mechanism must be able to accept a maximum-size Pup; if the actual network cannot directly encapsulate a packet of that size for transmission, the driver must include some form of intranetwork fragmentation.

A network driver may also be asked to broadcast a packet to all other hosts on that net. On some networks this is straight-

forward; on others it may require use of a reverse-path forwarding algorithm [6] or brute-force replication of the packet to each destination.

The transport mechanisms do not have to be perfectly reliable, but they should be successful most of the time—a packet loss rate of less than 1 percent is usually acceptable. A network operating for a short time in a degraded mode with a higher loss rate is harmless, so long as the probability is low that Pups will transit more than one net that is in this condition. However, if a network's inherent error characteristics are unfavorable, the driver should take steps to improve its performance, perhaps by incorporating a network-specific low-level acknowledgment and retransmission protocol.

To date, there have been five major types of networks integrated into the Pup architecture, each with a different level 0 driver.

Ethernet: Local Ethernet facilities can very easily serve as transport mechanisms for Pups: a Pup fits in an Ethernet packet with only a few additional words of encapsulation (see Fig. 2), and requires no fragmentation. These systems have good reliability, high speed, and can send broadcast packets [15], [21], [22].

MCA: The Multiprocessor Communications Adapter (MCA), a parallel TDM bus, serves as a local network tying together a limited number of Nova computers [7]. It has good reliability and requires no fragmentation, but does not support broadcast packets. Broadcasts are accomplished by the brute-force method, sending a copy of a broadcast packet to each of the possible hosts.

ARPANET: To cover longer distances, Pups can be routed through the ARPANET; the format for encapsulating a Pup in an ARPANET message is shown in Fig. 2. (Note that ARPANET Pup transport is based on host-IMP protocol messages, not on host-host protocol streams.) Because the standard maximum Pup length is less than that of an ARPANET message, the driver itself need not fragment Pups; however, the ARPANET does perform network-specific fragmentation internally: one "message" containing a Pup may become multiple "packets" within the ARPANET. Furthermore, the ARPANET provides increased reliability through the use of its own internal acknowledgment and retransmission protocols. The ARPANET does not presently support broadcast packets; rather than sending packets to all possible ARPANET hosts, the network driver does not implement broadcasts at all.

Leased Line Store-and-Forward Network: More frequently, different local networks are interconnected over long distances through the use of a private store-and-forward network constructed using leased telephone circuits. Similar in spirit to the ARPANET, this system is used to connect internetwork gateways. Unlike the ARPANET, the system does not use separate packet switches (IMP's), but instead switches packets through the hosts themselves; that is, the connected hosts include network-specific drivers that implement a store-and-forward network. This network has its own adaptive routing procedure, independent of the internetwork routing. The system is fairly reliable and does not require low-level acknowledgments. At present, the network drivers do not fragment Pups, but they do promote small packets to the front of transmission queues at intermediate points to help improve performance for interactive traffic.

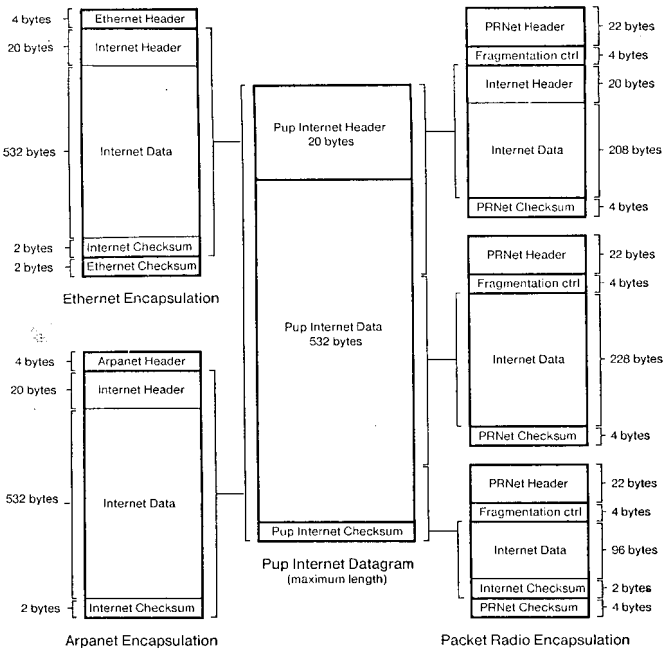

Fig. 2. Pup encapsulation in various networks.

Packet Radio Network: On an experimental basis, the ARPA packet radio network [10] has been used to carry traffic among local networks in the San Francisco Bay area. The packet radio network was integrated into the system by building a suitable level 0 network driver [23]. The system provides good reliability; but due to the relatively small maximum packet size (232 bytes), the driver must perform fragmentation and reassembly (see Fig. 2). Though using a broadcast medium, the packet radio protocols do not support broadcast packets. In this case, the low-level driver includes a procedure to periodically identify packet radio hosts that might be running Pup software; when asked to broadcast a packet, the driver sends copies of it to all such hosts.

To date we have not used any public packet-switched networks, such as Telenet, as packet transport mechanisms. These systems usually provide only a virtual circuit interface (X.25) that requires a user to pay for functionality that may not be needed. Compared to our existing leased line network, a Telenet-based packet transport mechanism would not be cost-effective except under conditions of very light traffic volume. We would prefer to use a service that provided simple, unreliable datagrams; if there were an appropriate interface, we could dismantle our leased line store-and-forward network.

B. Level 1: Internetwork Datagrams

This is the level at which packet formats and internetwork addresses are standardized. It is the lowest level of process-to-process communication.

1) *Pup Format*: The standard format for a Pup is shown in Fig. 3. The following paragraphs highlight the sorts of information required at the internet datagram level.

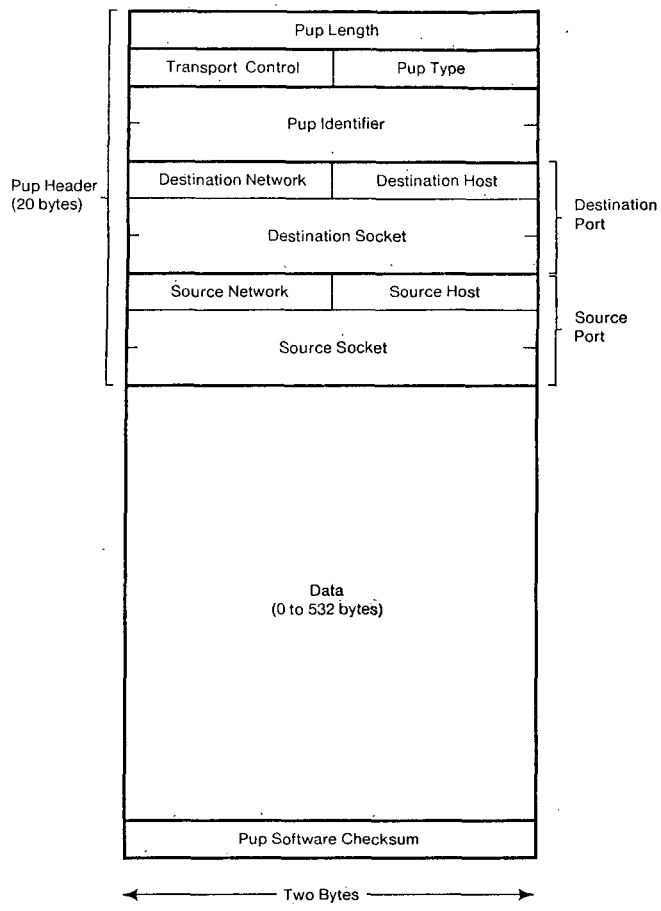

Fig. 3. The Pup internet datagram.

The *Pup length* is the number of 8-bit bytes in the Pup, including internetwork header (20 bytes), contents, and checksum (2 bytes).

The *transport control* field is used for two purposes: as a scratch area for use by gateways and as a way for source processes to tell the internet how to handle the packet. (Other networks call this the "facilities" or "options" field.) The *hop count* subfield is incremented each time the packet is forwarded by a gateway. If this ever overflows, the packet is presumed to be traveling in a loop and is discarded. A *trace bit* is specified, for potential use in monitoring the path taken by a packet.

The *Pup type* is assigned by the source process for interpretation by the destination process and defines the format of the Pup contents. The 256 possible types are divided into two groups. Some types are *registered* and have a single meaning across all protocols; Pups generated or interpreted within the internet (e.g., by gateways) have types assigned in this space. Interpretation of the remaining *unregistered* types is strictly a matter of agreement between the source and destination processes.

The *Pup identifier* is used by most protocols to hold a sequence number. Its presence in the internetwork header is to permit a response generated within the internet (e.g., error or trace information) to identify the Pup that triggered it in a manner that does not depend on knowledge of the higher level protocols used by the end processes.

Pups contain two addresses: a *source port* and a *destination port.* These hierarchical addresses include an 8-bit network number, an 8-bit host number, and a 32-bit socket number. Hosts are expected to know their own host addresses, to discover their network numbers by locating a gateway and asking for this information, and to assign socket numbers in some systematic way not legislated by the internet protocol.

There are some important conventions associated with the use of network addresses. A distinguished value of the network number field refers to "this network" without identifying it. Such a capability is necessary for host initialization (since most hosts have no permanent local storage and consequently no *a priori* knowledge of the connected network number), and to permit communication to take place within a degenerate internet consisting of an unidentified local network with no gateways. A distinguished value of the destination host address is used to request a broadcast. Certain values of the socket number field refer, by convention, to "well-known sockets" associated with standard, widely-used services, as is done in the ARPANET.

The *data* field contains up to 532 data bytes. The selection of a standard maximum packet length must reflect many considerations: error rates, buffer requirements, and needs of specific applications. A reasonable value might range anywhere from 100 to 4000 bytes. In practice, much of the internet traffic consists of packets containing individual "pages" of 512 bytes each, reflecting the quantization of memory in most of our computers. But just carrying the data is not enough, since the packet should accommodate higher level protocol overhead and some identifying information as well. Allowing 20 additional bytes for such purposes, we arrive at 532 bytes as the maximum size of the data field (a somewhat unconventional value in that it is not a power of two). Thus, there may be between 0 and 532 content bytes in a Pup, so its total length will range from 22 to 554 bytes. Pups longer than 554 bytes are not prohibited and may be carried by some networks, but no internetwork gateway is required to handle larger ones.

The optional *software checksum* is used for complete end-to-end coverage—it is computed as close to the source of the data and checked as close to the ultimate destination as is possible. This checksum protects a Pup when it is not covered by some network-specific technique, such as when it is sitting in a gateway's memory or passing through a parallel I/O path. Most networks employ some sort of error checking on the serial parts of the channel, but parallel data paths in the interface and the I/O system often are not checked.

The checksum algorithm is intended to be straightforward to implement in software; it also allows incremental updating so that intermediate agents which modify a packet (gateways updating the hop count field, for example) can quickly update the checksum rather than recomputing it. The checksum may (but need not) be checked anywhere along a Pup's route in order to monitor the internet's integrity.

2) *Routing*: Accompanying the packet format defined at level 1 are the protocols for internetwork routing. Each host, whether or not it is a gateway, executes a routing procedure on every outgoing Pup, as illustrated in Fig. 4. This procedure

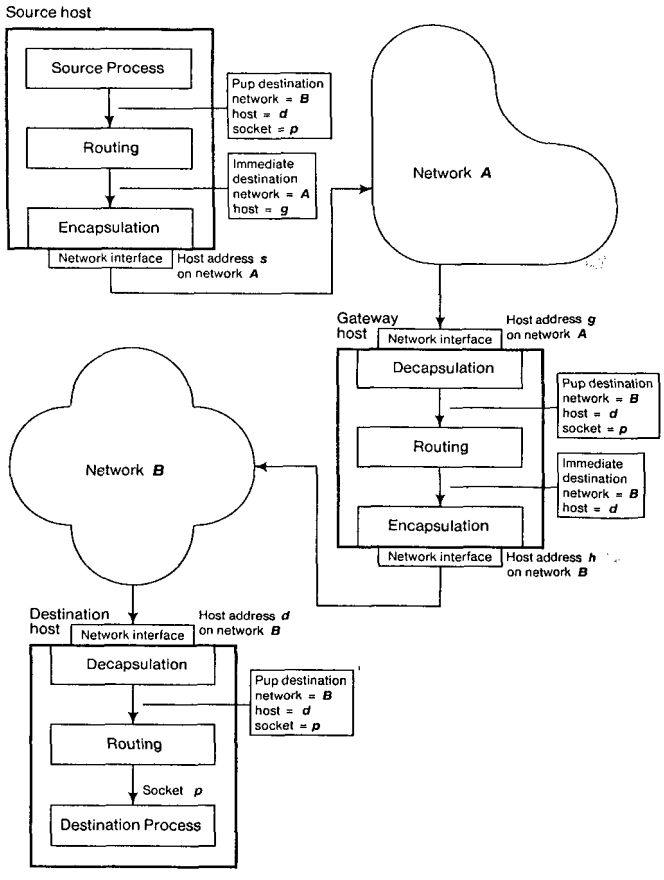

Fig. 4. Internetwork routing.

decides, as a function of the Pup destination port field, upon which *directly-connected network* the Pup is to be transmitted (if there is more than one choice), and it yields an *immediate destination host* which is the address on that network of either the ultimate destination or some gateway believed to be closer to the destination. Each routing step employs the same algorithm based on local routing information, and each Pup is routed independently.

Routing information is maintained in a manner very similar to the ARPANET-style adaptive procedures [12]. The initial metric used for selecting routes is the "hop count," the number of intermediate networks between source and destination. The protocol for updating the routing tables involves exchanging Pups with neighboring gateways and rests logically at level 2 of the protocol hierarchy. This is an example of a connectionless protocol which does not require perfectly reliable transmission for correct operation. Changes in internetwork topology may cause different gateways' routing tables to become momentarily inconsistent, but the algorithm is stable in that the routing tables rapidly converge to a consistent state and remain that way until another change in topology occurs.

A host which is not a gateway still implements a portion of this level 2 routing update protocol: it initially obtains an internetwork routing table from a gateway on its directly-connected network, and it obtains updated information periodically. If there is more than one gateway providing connections to other networks, the host can merge their routing tables and thus be able to select the best route for packets directed to any network.

C. Level 2: Interprocess Communication

Given the raw datagram facility provided at level 1, we can begin to build data transport protocols, tailored to provide appropriate levels of reliability or functionality for real applications.

These protocols generally fall into two categories: those in which a connection is established for a sustained exchange of packets, and those in which individual packets are exchanged on a connectionless basis. Connection-style protocols usually transport data very reliably, and transparently.

EFTP–The Easy File Transfer Protocol: This is a very simple protocol for sending files. Each data Pup gives rise to an immediate acknowledgment, and there is at most one Pup outstanding at a time. This protocol is an indirect descendant of the one outlined in [15]. Its simplicity makes this piece of communication mechanism easy to include under conditions of very limited resources. For example, we have implemented a complete EFTP receiver in 256 words of assembly language, for use in a network-based bootstrap and down-line loading process.

Rendezvous and Termination Protocol (RTP): This is a general means to initiate, manage, and terminate connections in a reliable fashion [28]. In normal use, an RTP user initiates a connection by communicating with a well-known socket at some server. That server will spawn a new port to actually provide the service, and the RTP will establish contact with this port. It employs a nonreusable *connection identifier* to distinguish among multiple instantiations of the same connection and to cope with delayed packets without making assumptions about maximum packet lifetimes. RTP also synchronizes Pup identifiers for use in managing the connection.

Byte Stream Protocol (BSP): This is a relatively sophisticated protocol for supporting reliable, sequenced streams of data. It provides for multiple outstanding packets from the source, and uses a moving window flow control procedure. User processes can place *mark bytes* in the stream to identify logical boundaries and can send out-of-band *interrupt* signals. RTP and BSP combined perform a function similar to that of the TCP, with which they share a certain degree of common ancestry [1], [17].

Connectionless protocols do not attempt to maintain any long-term state; they usually do not guarantee reliability, but leave it up to the designer to construct the most suitable system. Their simplicity and ease of implementation make them extremely useful.

Echo: A very simple protocol can be used to send test Pups to an *echo server* process, which will check them and send back a reply. Such servers are usually embedded in gateways and other server hosts, to aid in network monitoring and maintenance. The server is trivial to implement on top of the level 1 facilities.

Name Lookup: Another server provides the mapping from string names of resources to internetwork addresses; this is accomplished by a single exchange of packets. This service is

often addressed with a broadcast Pup, since it is used as the first step in locating resources. (The name lookup service itself, of course, must be located at a well-known address. To be useful, it must be widely available; therefore, it is typically replicated at least once per network.)

Routing Table Maintenance: The internetwork routing tables are maintained by Pup's exchanged periodically among internetwork gateways and broadcast for use by other hosts.

Page-Level File Access: The Woodstock file server (WFS), one of the family of file servers available on the internet, provides page-at-a-time access to a large file store [29]. The protocols used for this do not require establishment of a connection, but merely exchange request and response Pups that each carry both commands and file data. This arrangement supports random-access, transaction-oriented interactions of very high performance, frequently better than that obtained using local file storage, because the file server's disks are much faster than those typically connected to personal computers.

Gateway Monitoring and Control: There is no single network control center, but individual gateways may be queried from a monitoring program run on any user machine. With suitable authentication, the user may assume remote control of the gateway so as to perform operations such as changing parameters and loading new versions of the software.

Other connectionless protocols are used to access a *date and time server*, an *authentication server*, and a *mail check server* integrated with an on-line message system. These protocols are designed to be as cheap as possible to implement (i.e., without connection overhead) so that such servers may be replicated extensively and accessed routinely without consuming excessive resources. For example, instances of some of these servers are present in all gateway hosts so as to maximize their availability.

D. Level 3: Application Protocols

Armed with a reasonable collection of data transport protocols at level 2, one can begin to evolve specific applications at level 3. These are supported by various function-oriented protocols [4].

Telnet: Terminal access to remote hosts is provided with an internetwork Telnet protocol, which makes use of the combination of the rendezvous and termination protocol (RTP) and the byte stream protocol (BSP) at level 2. Using the notion of a virtual terminal, Telnet implementations map characteristics of actual terminals into a network-independent representation; a mark byte in the stream and an out-of-band interrupt, for example, are used to signal an "attention." (This approach is a subset of the ARPANET Telnet protocol, without any of its options such as RCTE [8], [9].)

FTP: The RTP and BSP are again combined as the foundation for an internetwork file transfer protocol (FTP), supporting stream-oriented access to files. The underlying byte streams provide reliable communication, and the major task of FTP is to communicate commands and responses and to sort out different representations of data in different file systems. FTP implementations have been embedded within existing time-sharing systems, and also constitute the core of dedicated, high-capacity file servers.

Printing: Among the important shared resources in the internet are high-quality printing servers. Rather than using the fully developed BSP and FTP, the specialized task of sending unnamed, standard format document files to a printer makes use of the more restricted but much simpler EFTP.

CopyDisk: Given high-performance networks and simple gateways that can forward Pups among them efficiently, it is perfectly reasonable to copy entire disk packs through the internet. The CopyDisk protocol negotiates between the participating machines to ensure that the disks are compatible, and handles error recovery should something break down.

Remote Graphics: Personal display-oriented computers such as the Alto can be used to provide a convivial front end for large programming systems such as Interlisp. The Alto Display protocol is used for exchanging descriptions of graphical structures as well as text; it is similar to the ARPA network graphics protocol, but with extensions to support raster-scanned graphics [24], [25], [30].

Additional applications have included cooperative editing of common documents from multiple machines, audio communication and packet voice, and many others.

As users create new applications, these systems tend to develop their own natural layering of function. Some may require new protocol designs in the existing hierarchy; the Pup architecture permits this degree of flexibility down to the level of the simple internetwork datagram. As we gain experience with new systems, common pieces of design will begin to emerge that might be of more general use; they will eventually find their way into an appropriate place in this hierarchy of communications protocols.

IV. EVOLUTION, ACTUAL EXPERIENCE, AND PERFORMANCE

The Pup architecture emerged against a background of ARPANET protocols. Many of its important ideas—and those of its key relative, TCP—first appeared during the course of a series of meeting of the International Network Working Group (IFIP TC-6 WG6.1) during 1973. Pup and TCP share a number of important principles, most notably that of reliable end-to-end transmission through an internet. Pup subsequently diverged from TCP as the desire for implementation within Xerox required decoupling it from TCP's long and sometimes painful standardization process.

The fundamentals of the Pup design crystallized in 1974 and have remained essentially unchanged since then. During this interval many higher level protocols have been developed, the implementations have evolved considerably, and the internetwork system has grown to include approximately 1000 hosts, attached to 25 networks of 5 different types, using 20 internetwork gateways. The system is in regular use, is quite stable, and requires little regular maintenance or attention.

From a functional point of view, this internetwork architecture has been able to fulfill the needs of a very diverse community. While the bulk of all traffic is carried by means of a few standard protocols, it has proven extremely valuable

to be able to define new protocols—aiming at different points in the space of performance, cost, and functionality—and to fit them into the internet protocol hierarchy at any of several levels.

In terms of performance, the internetwork gateways impose very little overhead because they are so simple. In regions of the internet where multiple high-bandwidth local networks are interconnected directly by a single minicomputer-based gateway, there is almost no noticeable difference between intranet and internet performance. Total throughput in an individual gateway is high, ranging from 400 to 1000 kbits/s (depending on the particular implementation), and the typical delay experienced by maximum-length Pups in the case just mentioned is 2 to 5 ms.

These figures do not represent limits to what is achievable, even with the relatively low-powered machines now being used as gateways, because the gateway software has not been highly tuned for this application but rather is based on general-purpose software packages that are also used in many other hosts. But the current performance is adequate because the internetwork traffic load is typically only a tiny fraction of the capacity of the underlying local network channels. There exists one Alto-based gateway that interconnects three 3-Mbit/s Ethernet channels as well as several 9.6-kbit/s leased lines and a packet radio interface. In general the bottlenecks are not the gateways but rather the slower communication channels; discard of Pups due to congestion in gateways is almost exclusively due to overload of the 9.6-kbit/s lines.

As might be expected, most of the traffic in our local networks is intranetwork, that is, consisting of Pups whose source and destination are on the same network. For example, measurement of one such network has shown a typical volume of 2.2 million packets per day, 72 percent of which are intranetwork packets [22]. Furthermore, of the remaining 28 percent, more than half consist of traffic to or from another nearby local network connected via a single gateway. (This site is served by multiple local networks because it is too large to cover with a single one using existing Ethernet technology, and also because it would exhaust a single network's address space.) The rest of the traffic—some 250 000 packets per day—is transported to or from other campuses in the internet, mostly via the leased line network.

The higher level protocols, such as the byte stream and FTP, are generally limited in performance by the processor capacity or the secondary storage bandwidth at the source and destination. For example, our BCPL implementation of BSP can maintain a data stream at the rate of about 500 kbits/s between end processes running on Alto minicomputers, at which point both machines are CPU-bound. While it is certainly adequate for most applications, we find this performance somewhat disappointing, and we view it as an indication that BSP—although substantially simpler than, say, TCP—is still too complicated a protocol for high-performance communication.

The Pup architecture allows individual networks to be added to the internet system on an ad hoc basis, with no need for central control or coordination except to assign new network numbers. Users sharing a local network can assemble gateways and lease lines to other nearby gateways; they are encouraged to make multiple intergateway connections to provide alternate routes and thereby reduce the probability of being isolated. The gateway software has evolved to the point where if one starts a copy of it on a host having at least one connection to the existing internet, it will automatically obtain the files and other information it needs, announce its availability to the rest of the internet, and begin forwarding Pups.

V. A RETROSPECTIVE CRITIQUE, POSSIBLE IMPROVEMENTS, AND FUTURE RESEARCH

While the architecture works extremely well, there are some lessons to be learned from this experience.

A. Addressing and Routing

The size of address fields is a question of continuing controversy. An 8-bit network number supports up to 256 nets; that is fine for now, but eventually it should be made larger. To date, 256 hosts per net has not been a problem, though it is likely to become one (for example, when the ARPANET's new 24-bit addressing convention starts to receive wide use). We have avoided variable-length address fields in the Pup design because they increase per-packet processing costs.

If an internetwork system becomes extremely large, the number of networks becomes so great that it is no longer practical for all hosts to keep routing table entries for all possible destination networks. Area routing strategies may be employed to attack this problem [12]. Alternatively, one may adopt a scheme in which the local routing table becomes a cache of recently used routing information, with routes to specific networks computed and maintained as needed. The problem of locating routes to distant parts of the internet is an area of current research.

One could consider revising the entire notion of a hierarchical address space. Under the current design, it is sometimes necessary to change the host number of a machine which is moved from one net to another—an operational annoyance. It is conceivable that every host could be given a unique address within a flat address space; a more sophisticated mechanism would then be needed to map addresses into routes, since there would no longer be a network number as part of the address (except perhaps as a hint, to improve performance).

We view with some disfavor nonhierarchical organizations in which internet addresses consist of a concatenation of network-specific addresses [27]. Such arrangements have the effect of fixing the *path* to a given destination and blur the distinction between addressing and routing.

Socket numbers, which are now 32 bits wide, could easily shrink to 16. Originally, 32 bits were assigned to allow inclusion of a unique subfield to distinguish among multiple instantiations of a connection; we now recognize that a better approach is to use a distinct connection identifier at the time of connection is established, as mentioned earlier in the presentation of the rendezvous and termination protocol.

Using hop counts as the metric for routing decisions has worked remarkably well. An obvious drawback, however, is

that it considers a hop through a 9.6-kbit/s phone line equally as good as a hop through a 3-Mbit/s Ethernet link. As the topology becomes more richly connected, this will increasingly become a problem. We intend eventually to change the routing algorithms to reflect some consideration of bandwidth and delay, using known techniques based on research into adaptive distributed routing algorithms in the ARPANET and elsewhere.

We have given little consideration to source routing or other forms of advice (e.g., class of service) provided to the internet routing procedures by source processes. In providing such facilities, one must take great care not to compromise the simplicity of the basic internet datagrams or violate the layering of protocols.

B. Congestion Control and Utilization of Low-Bandwidth Channels

The current congestion control techniques must be regarded as primitive. Discarding Pups and (where possible) notifying the source process when congestion occurs has the virtue of simplicity, and we believe it is a good general approach; but the present design has several defects. Insufficient information is returned to the source process to enable it to make an informed decision about how to proceed; further, the discard of Pups is haphazard, and no provision is made for fairness. Congestion occurs most often at the entry to slow channels, and under overload conditions the perceived performance of paths through those channels is highly variable.

This is a situation in which it would be appropriate to perform a relatively large amount of computation per packet in order to optimize the utilization of the communication bandwidth. For example, the network-specific driver for a leased telephone circuit could examine the source and destination addresses of Pups to deduce the existence of "conversations," and use this information to share the slow channel more effectively. (The Arpanet IMP's deduce conversations in precisely this way, though for purposes having to do primarily with flow control rather than congestion control.)

In the same vein, techniques such as code compression, elimination and regeneration of identical internet headers in successive packets, etc., may be implemented in the network-specific drivers for the slow channels, with minimal impact on the end-to-end protocols. Such techniques are used widely in virtual circuit designs, and their applicability is sometimes cited as an advantage of virtual circuits over datagrams [18]. But there is no reason they cannot be employed in a datagram-based internet, so long as the necessary additional computation is done in the right place.

The important point is that optimizing the utilization of the communication channel is appropriate only when the channel bandwidth is scarce compared to the computation required to perform such optimization. Where the processing capacity of the end machines is itself the scarce resource, as we have observed in the local network environment, such techniques are highly inappropriate.

C. Pup Types in the Internet Header

The distinction between registered and unregistered Pup types at the level of internet datagrams has not turned out to be particularly useful, except in a few cases: Pups of type "error" and "trace" may be generated from within the internet without knowledge of the higher level protocols being employed by the end processes.

D. Performance of Reliable End-to-End Protocols

Present implementations of the byte stream protocol include fairly sophisticated adaptive flow control heuristics that also try to take note of any packets lost due to internet congestion. This approach has worked reasonably well in enabling a source to adapt to the conditions encountered along the path to a particular destination. However, use of networks with highly variable behavior, such as the wide-ranging delays experienced when using the packet radio network, can confound these heuristics. Under unusual circumstances, the flow control procedures have been observed to move suddenly into very unfavorable operating regions. The difficulty involving the radionet has since been solved, but the general design of simple, effective flow control and congestion control procedures is just a very hard problem, particularly procedures intended to adapt dynamically to and make good use of different networks whose performance may vary by nearly three orders of magnitude.

The step from raw Pups to a byte stream may be too large. The byte stream protocol does too much for many applications; it is complex enough that few systems have ever implemented the entire specification. As discussed previously, performance of the BSP, when compared to some other systems, is reasonable; but it does not give a user the full capacity of the underlying networks. In a high-bandwidth local network environment, paying attention to per-packet processing overhead is of extreme importance.

We have considered, but have not yet implemented, a proposal for an intermediate level of functionality: a reliable packet protocol (RPP) that takes care of connection establishment and processes flow control information, but tries not to dictate how a client program should do buffer management. It ensures reliable delivery (i.e., each packet once and only once), but may deliver packets to the client out of order, and does not deliberately attempt to hide packet boundaries. A BSP connection, where that is what is desired, may then be reimplemented as a veneer on top of an RPP connection.

E. Access to the Internet

The present Pup architecture can be characterized as "open": users and applications are permitted, and indeed encouraged, to take advantage of the internet for routine communication. Access to the internet is uncontrolled; as in many network designs, responsibility for access control rests with the host systems, and whatever accounting is performed is for the services rendered by individual servers. In our research and development environment this is ideal, but obviously in some other environments it might not be.

F. Conclusions

The success of Pup as an internetwork architecture depends on a number of important principles. Key among these is the

layering of function in such a way that applications may make use of the internet at any of several levels, with the ability to choose among alternative protocols at each level or to develop new ones where necessary. Simple internetwork datagrams constitute the level at which media independence (through encapsulation) is achieved; they are also the unit of direct process-to-process communication. This is crucial both to flexibility and to performance, particularly in an internetwork environment dominated by relatively lightweight hosts and high-bandwidth local networks.

During 1976, the Pup internet reached a level of functionality roughly equivalent to that provided by the standard ARPANET protocols—byte streams, Telnet, and FTP. From that time to the present we have concentrated on building servers and constructing applications to access them through the internet. We are just beginning to explore that area of interprocess communication traditionally considered the domain of multiprocessors. Some interesting opportunities arise from the availability of 100 or so minicomputers interconnected by a 3-Mbit/s broadcast channel, and by ten or so similar clusters, all interconnected by a store-and-forward network. We believe that the Pup architecture serves as a good foundation for such investigations.

ACKNOWLEDGMENT

A large systems effort such as the development of Pup reflects the efforts of many different participants. Other people who have implemented parts of Pup and contributed ideas include W. Crowther, Y. Dalal, H. Murray, B. Sproull, L. Stewart, J. White, and G. Williams.

We also wish to thank D. Cohen, D. Crocker, B. Kahn, J. Postel, and C. Sunshine for their careful reading of an earlier draft of this paper.

REFERENCES

[1] V. G. Cerf and R. E. Kahn, "A protocol for packet network intercommunication," *IEEE Trans. Commun.*, vol. COM-22, pp. 637–648, May 1974.

[2] V. G. Cerf and P. T. Kirstein, "Issues in packet-network interconnection," *Proc. IEEE*, vol. 66, pp. 1386–1408, Nov. 1978.

[3] D. Cohen, "Issues in transnet packetized voice communication," presented at the 5th Data Commun. Symp., Snowbird, UT, Sept. 1977.

[4] S. D. Crocker, J. F. Heafner, R. M. Metcalfe, and J. B. Postel, "Function-oriented protocols for the ARPA computer network," in *AFIPS Conf. Proc. Spring Joint Comput. Conf.*, vol. 40, 1972.

[5] Y. K. Dalal, "Broadcast protocols in packet switched computer networks," Stanford Univ. Digital Syst. Lab., Tech. Rep. 128, Stanford, CA, Apr. 1977.

[6] Y. K. Dalal and R. M. Metcalfe, "Reverse path forwarding of broadcast packets," *Commun. Ass. Comput. Mach.*, vol. 21 Dec. 1978.

[7] Data General Corp., "Type 4038 multiprocessor communications adapter," Tech. Ref. 014-000002-01, Sept. 1971.

[8] J. Davidson, W. Hathaway, J. Postel, N. Mimno, R. Thomas, and D. Walden, "The Arpanet Telnet protocol: Its purpose, principles, implementation, and impact on host operating system design," in *Proc. 5th Data Commun. Symp.*, Snowbird, UT, Sept. 1977.

[9] E. Feinler and J. Postel, Eds., "Telnet protocol specification," in *Arpanet Protocol Handbook*, Jan. 1978.

[10] R. E. Kahn, S. A. Gronemeyer, J. Burchfiel, and R. C. Kunzelman, "Advances in packet radio technology," *Proc. IEEE*, vol. 66, pp. 1468–1496, Nov. 1978.

[11] A. C. Kay, "Microelectronics and the personal computer," *Sci. Amer.*, vol. 237, Sept. 1977.

[12] J. M. McQuillan "Adaptive routing algorithms for distributed computer networks," Ph.D. dissertation, Harvard Univ., Cambridge, MA, 2831, Bolt Beranek and Newman, Rep. 2831, May 1974.

[13] J. M. McQuillan and D. C. Walden, "The ARPANET design decisions," *Comput. Networks*, vol. 1, Aug. 1977.

[14] R. M. Metcalfe, "Packet communication," Ph.D. dissertation, Harvard Univ., Cambridge, MA, M.I.T. Project Mac TR-114, Dec. 1973.

[15] R. Metcalfe and D. Boggs, "Ethernet: Distributed packet switching for local computer networks," *Comm. Ass. Comput. Mach.*, vol. 19, July 1976.

[16] R. Needham and M. Schroeder, "Using encryption for authentication in large networks of computers," *Comm. Ass. Comput. Mach.*, vol. 21, Dec. 1978.

[17] J. Postel, "Internetwork protocols," this issue, pp. 604–611.

[18] L. G. Roberts, "The evolution of packet switching," *Proc. IEEE*, vol. 66, pp. 1307–1313, Nov. 1978.

[19] J. F. Shoch, "Internetwork naming, addressing, and routing," in *Proc. 17th IEEE Comput. Soc. Int. Conf. (CompCon)*, Sept. 1978.

[20] ——"Packet fragmentation in internetwork protocols," *Comput. Networks*, vol. 3, Feb. 1979.

[21] ——"Design and performance of local computer networks," Ph.D. dissertation, Stanford Univ., Stanford, CA, University Microfilms, Aug. 1979.

[22] J. F. Shoch and J. A. Hupp, "Performance of an Ethernet local network—A preliminary report," in *Proc. Local Area Network Symp.*, Boston, MA, May 1979.

[23] J. F. Shoch and L. Stewart, "Interconnecting local networks via the packet radio network," in *Proc. 6th Data Comm. Symp.*, Pacific Grove, CA, Nov. 1979.

[24] R. F. Sproull and D. Cohen, "High-level protocols," *Proc. IEEE*, vol. 66, pp. 1371–1386, Nov. 1978.

[25] R. Sproull and E. Thomas, "A network graphics protocol," *Comput. Graphics*, vol. 8, Fall 1974.

[26] C. Sunshine, "Interconnection of computer networks," *Comput. Networks*, vol. 1, Jan. 1977.

[27] C. Sunshine, "Source routing in computer networks," *ACM Comput. Commun. Rev.*, vol. 7, Jan. 1977.

[28] C. Sunshine and Y. Dalal, "Connection management in transport protocols," *Comput. Networks*, vol. 2, Dec. 1978.

[29] D. Swinehart, G. McDaniel, and D. Boggs, "WFS: A simple shared file system for a distributed environment," *Oper. Syst. Rev.*, vol. 13, Nov. 1979.

[30] W. Teitelman, "A display-oriented programmer's assistant," in *Proc. 5th Int. Joint Conf. on Artificial Intelligence*, Cambridge, MA, Aug. 1977; also available as Xerox PARC Tech. Rep. CSL-77-3.

[31] C. P. Thacker, E. M. McCreight, B. W. Lampson, R. F. Sproull, and D. R. Boggs, "Alto: A personal computer," *Computer Structures: Readings and Examples*, Siewiorek, Bell, and Newell, Eds., 1980.

[32] H. Zimmermann, "The ISO reference model," this issue, pp. 425–432.

★

David R. Boggs (S'69–M'75) received the B.S.E.E. degree from Princeton University, Princeton, NJ, and the M.S.E.E. degree from Stanford University, Stanford, CA.

He is presently a Ph.D. candidate at Stanford University. Since 1973 he has been a member of the research staff at the Xerox Palo Alto Research Center, Palo Alto, CA, working in the area of computer communication and distributed computing.

Mr. Boggs is a member of the Association for Computing Machinery and holds amateur and commercial radio licenses.

★

John F. Shoch (M'76) received the B.A. degree in political science and the M.S. and Ph.D. degrees in computer science from Stanford University, Stanford, CA.

Since 1971 he has been a member of the research staff at the Xerox Palo Alto Research Center, Palo Alto, CA. Research interests have included communications protocols, local computer networks, internetworking, distributed systems, and programming language development. He has taught at Stanford University.

Dr. Shoch is a member of the Association for Computing Machinery and

serves as Vice-Chairman (United States) of IFIP Working Group 6.4 on Local Computer Networks.

★

Edward A. Taft received the B.A. degree in applied mathematics from Harvard University, Cambridge, MA, in 1973.

Since then he has been a member of the Computer Science Laboratory of the Xerox Palo Alto Research Center, Palo Alto, CA, working in the areas of internetwork protocols, distributed systems, and personal computing.

Robert M. Metcalfe received the S.B. degree in electrical engineering and the S.B. degree in management from the Massachusetts Institute of Technology, Cambridge, in 1969, and the M.S. degree in applied mathematics and the Ph.D. degree in computer science from Harvard University, Cambridge, MA, in 1970 and 1973, respectively.

His Ph.D. dissertation is titled "Packet Communication." He is presently President of 3Com Corporation and Consulting Associate Professor of Electrical Engineering at Stanford University, Stanford, CA, where he has been lecturing on distributed computing since 1975. He was with Xerox Corporation, Palo Alto, CA, between 1972 and 1979, where he worked on ARPANET, Ethernet, Fibernet, Pup, and Laurel. In June 1979 he formed 3Com Corporation to promote, develop, and exploit communication compatibility among computers in the office and home.

Formal Methods in Communication Protocol Design

GREGOR V. BOCHMANN AND CARL A. SUNSHINE

(Invited Paper).

Abstract—**While early protocol design efforts had to rely largely on seat-of-the-pants methods, a variety of more rigorous techniques have been developed recently. This paper surveys the formal methods being applied to the problems of protocol specification, verification, and implementation.**

In the specification area, both the service that a protocol layer provides to its users and the internal operations of the entities that compose the layer must be defined. Verification then consists of a demonstration that the layer will meet its service specification and that each of the components is correctly implemented. Formal methods for accomplishing these tasks are discussed, including state transition models, program verification, symbolic execution, and design rules.

I. INTRODUCTION

As evidenced by the earlier papers of this issue, increasingly numerous and complex communication protocols are being employed in distributed systems and computer networks of various types. The informal techniques used to design these protocols have been largely successful, but have also yielded a disturbing number of errors or unexpected and undesirable behavior in most protocols. This paper describes some of the more formal techniques which are being developed to facilitate design of correct protocols.

Manuscript received August 8, 1979; revised January 8, 1980. This work was supported in part by the National Sciences and Engineering Council of Canada and the United States Advanced Research Projects Agency.

G. V. Bochmann is with the University of Montreal, Montreal, P.Q., Canada.

C. A. Sunshine is with the Information Sciences Institute, University of Southern California, Marina del Rey, CA 90221.

As they develop, protocols must be described for many purposes. Early descriptions provide a reference for cooperation among designers of different parts of a protocol system. The design must be checked for logical correctness. Then the protocol must be implemented, and if the protocol is in wide use, many different implementations may have to be checked for compliance with a standard. Although narrative descriptions and informal walk-throughs are invaluable elements of this process, painful experience has shown that by themselves they are inadequate.

In the following sections, we shall discuss the use of formal techniques in each of the major design steps of specification, verification, and implementation. Section II clarifies the meaning of specification in the context of a layered protocol architecture, identifies what a protocol specification should include, and describes the major approaches to protocol specification. Section III defines the meaning of verification, discusses what can be verified, and describes the main verification methods. Section IV provides pointers to some important case histories of the use of these techniques. For detailed examples, we refer to the subsequent papers of this issue which generally provide additional support for the points which we have had to treat briefly in this survey. A complete bibliography may be found in [18], and complementary surveys in [44], [8], [33], [43].

II. PROTOCOL SPECIFICATION

As noted above, protocol descriptions play a key role in all stages of protocol design. This section clarifies the meaning of specification in the domain of communication protocols,

0090-6778/80/0400-0624$00.75 © 1980 IEEE