IEN #140
April 1980

Mutual Encapsulation of Internetwork Protocols

by

John F. Shoch*, Danny Cohen†, and Edward A. Taft*

*Xerox Palo Alto Research Center, Palo Alto, California
†USC Information Sciences Institute, Marina del Rey, California

Abstract

Encapsulation is a key concept in developing a layered architecture for communication protocols: objects from one layer can be "encapsulated" for transmission through a lower layer in the architecture.

This approach can be used to combine different incompatible networks into a single "internetwork": packets from a unified internet protocol are encapsulated within the network-specific formats and protocols associated with each individual network.

But just as there are incompatible networks, we are now seeing the emergence of incompatible internetworks -- systems serving different user communities, yet perhaps overlapping in their use of particular networks. Full inter-operability of these systems would require some form of protocol translation, or else the development of an even higher level "inter-internetwork" protocol.

Yet different internetwork designs can combine some of their capabilities by invoking a higher level of encapsulation: each internetwork extends its range by using the other internetwork as one of its underlying communications links. This approach has been called "mutual encapsulation."

In this paper we explore the notion of mutual encapsulation; as an example we consider the coexistence of the Xerox internet environment with the Arpa internet environment. Each of these systems is composed of several individual networks with a variety of protocols and performance parameters; but with mutual encapsulation each can make use of the capabilities of the other.

1. Introduction

One of the common themes in the development of communications protocols has been the use of a *layered architecture* -- a well defined set of protocols stacked upon each other, with carefully specified interfaces between layers. In a packet-switched system, for example, an applications program may pass a block of data to a high-level protocol layer, which passes it down to a network control layer, which might pass it to the low-level network software. This layered modularity makes it easy to allocate particular functions to specific layers, hides unnecessary detail, and allows implementations at higher levels to remain independent of any lower level changes.

As a block of data or a packet is passed down through this hierarchy, suitable headers are usually added at each layer, in a process known as *encapsulation* (or *wrapping*). At its destination, a packet moves back up through the hierarchy and the appropriate headers are removed at each layer, in a process known as *decapsulation*. In addition to adding new header information, there is often a multiplexing function associated with the encapsulation procedure -- at each stage, packets from many different processes may be intermixed for handling in the next layer; similarly, there may then be a demultiplexing function associated with the decapsulation procedure. This notion of encapsulation has been used to define and implement protocol architectures for connecting many heterogeneous hosts to one network, as in the Arpanet.

Over time, however, we've seen the emergence of many different packet-switched networks -- alternative long-haul store-and-forward systems, packet radio networks, and local computer networks. With this increase has come the desire to support communication among

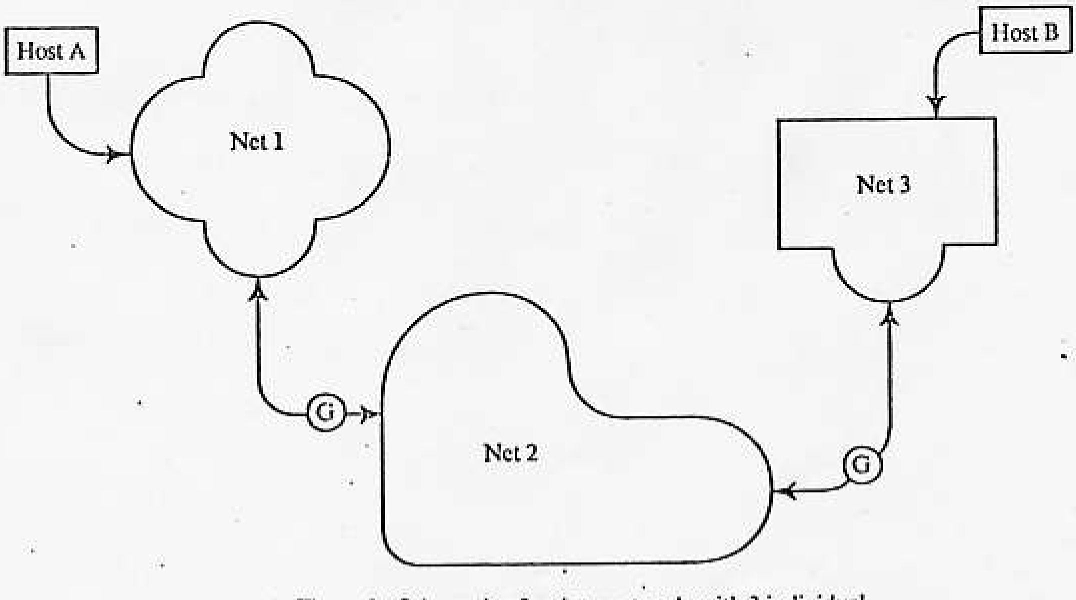

Figure 1: Schematic of an internetwork, with 3 individual networks and 2 internetwork gateways providing communication between Host A and Host B.

Note: This is a pre-print of a paper to be presented at the conference "Trends and Applications 1980: Computer Network Protocols," NBS, Gaithersburg, May 29, 1980.

machines connected to different networks, unifying these incompatible systems into an *internetwork* (often referred to as just an *internet*, or simply an IN) [Sunshine, 1977; Cerf & Kirstein, 1978]. This is usually accomplished with a layered architecture of *internetwork protocols*: an abstract *internet datagram* is defined, along with procedures to encapsulate these datagrams for transmission through individual networks (now often referred to as *packet transport mechanisms*). These individual networks are interconnected with *internetwork gateways* -- machines that are hosts on multiple networks -- which decapsulate a packet as it comes in off of one network, and encapsulate it for transmission through the next (see Figure 1). Note that the internet datagram serves as the basic level of commonality in the system: higher levels in the internet architecture are built up upon this layer, and thus are isolated from any network-dependent mechanisms (see Figure 2).

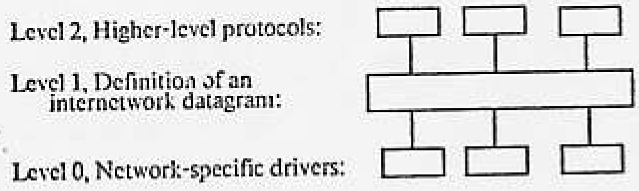
Level 2, Higher-level protocols:
Level 1, Definition of an internetwork datagram:
Level 0, Network-specific drivers:

Figure 2: Schematic of an internetwork architecture.

But just as we saw the development of incompatible networks there are now incompatible internetworks -- built at different times, with different objectives, or by different organizations. The basic model used is often similar (using internet datagrams and internetwork gateways), but the detailed designs of the internet protocols usually differ substantially. Fortunately, there are fewer internetworks then there are networks, but the desire still arises to share these resources.

One ultimate objective might be to provide full communication among all hosts connected to different internetwork systems. There are two basic approaches:

1. *Inter-internetwork protocols.* We could again recur on the basic design: define an even more general protocol architecture that would subsume the alternative internetwork systems, thus creating an inter-internetwork (or I^{2}N).

2. *Internetwork protocol translation gateways.* We have seen the manner in which simple internetwork gateways encapsulate packets for transmission through individual networks; a more sophisticated protocol translation gateway would attempt to map the protocols of one internetwork architecture directly into the protocols of an alternative design. This translation could take place at the packet level, or could be a translation of higher-level protocols.

At the moment, both of these approaches appear to be formidable efforts. An inter-internetwork protocol might have to encompass the union of all features in the various internet designs, and would require even more software development in end-user hosts. Furthermore, there may then be the inevitable development of alternative inter-internetwork schemes, thus requiring an even newer design of an I^{3}N, perhaps followed by an I^{4}N, and more.

A protocol translation gateway would have to dynamically map features from one design into another -- a process that does not always look feasible. (A limited form of protocol translation can take place at very high levels, however. A host that supports two independent internetwork architectures, for example, can be used to stage files in a file transfer -- the file is transferred in with one protocol, and transferred out with the other.)

While it may be difficult to provide direct inter-operability between different internet architectures, one would like to provide coexistence and mutual support between the two systems, and avoid duplication of facilities -- where possible. In particular, we would like to be able to share the use of the underlying communications networks, so that alternative architectures can extend their range of use. At the lowest level, two different internet architectures can share the use of a particular network, or packet transport mechanism. This may necessitate different forms of encapsulation and addressing, and produces two disjoint internets that overlap (see Figure 3.1).

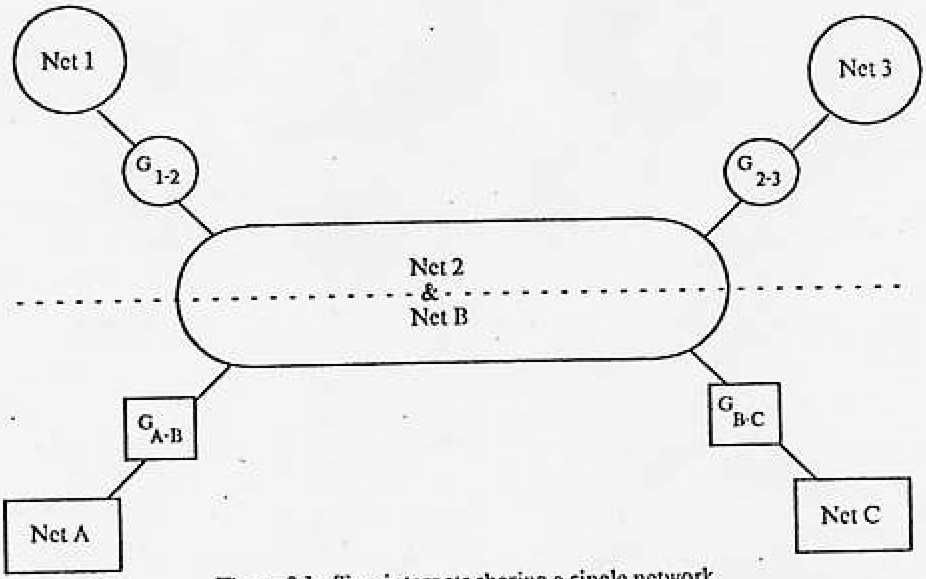

Figure 3.1: Two internets sharing a single network.

Unfortunately, this approach does not scale very gracefully: it requires gateways from both architectures between each pair of networks (see Figure 3.2).

There is, however, a more general approach: one internet can view the other internet system as one of its many packet transport mechanisms -- internet packets from one environment are merely encapsulated within the internet packets from the other system (see Figure 3.3). This does introduce an extra level of encapsulation.

Clearly this operation can take place in both directions: each internet system can view the other as one of its potential packet transport mechanisms, and they can mutually serve each other; the phrase *mutual encapsulation* is used to describe this mechanism.

This fundamental idea can have fairly wide applicability, in many contexts. To provide a bit more clarity in this discussion, however, we will examine the basic notion of mutual encapsulation by using a specific example -- interconnections between the Xerox internet environment and the Arpa internet environment. In the next section we review these two designs; in subsequent sections we explore in detail the mechanisms needed to provide mutual encapsulation.

2. The Xerox and Arpa internet activities

In the last several years two similar research efforts on internetworking have emerged: the Pup design at the Xerox Palo Alto Research Center and the Internet Protocol (IP) effort supported by Arpa. Both efforts trace their roots back to a series of meetings of the International Network Working Group (IFIP TC6 WG6.1, or INWG) during 1973. In the early years the two designs diverged in response to different research interests and requirements, and as a function of different design decisions. In recent years, however, the two approaches have shown a great deal of similarity.

The Pup design crystallized in 1974, and has grown to provide communications for over 1000 hosts, attached to 25 networks of 5 different types, using 20 internetwork gateways [Boggs, *et al.*, 1980]. The system now supports a wide range of applications, including file transfer, terminal access, network graphics, packet voice, and much more [Shoch, 1980].

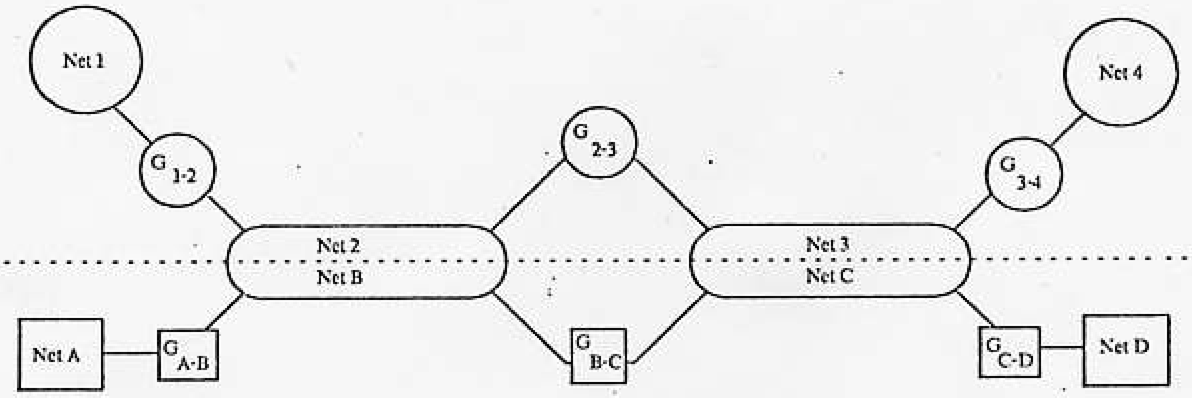

Figure 3.2: Inefficient sharing of multiple networks.

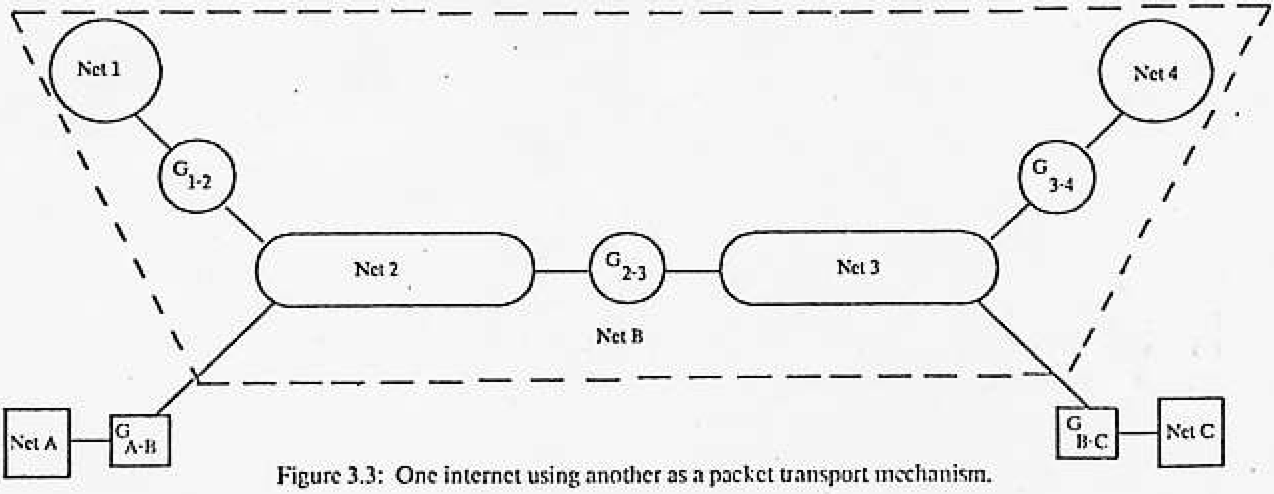

Figure 3.3: One internet using another as a packet transport mechanism.

Figure 4.1 shows a portion of the Pup architecture. At the bottom (level 0) are several different networks used to transport packets; in the current implementation these include various Ethernet local computer networks [Metcalfe & Boggs, 1976; Shoch & Hupp, 1979], the Arpanet, a private long-haul store-and-forward network, and the Bay Area Packet Radio Network [Kahn, *et al.*, 1978; Shoch & Stewart, 1979]. Level 1 defines the standard, abstract internetwork datagram, also known as a Pup: Pups provide datagram access to the internet. Above that layer we find the Byte Stream Protocol (BSP), which provides an error-controlled, flow-controlled stream of bytes. At level 3, we find several higher-level protocols, including the File Transfer Protocol (FTP), and the Telnet protocol used to support terminal connections.

Note that "Pup" is used in referring to three different entities: the overall Pup internet architecture, the Pup layer (level 1) in that architecture, and the Pup packets themselves.

The Arpa-supported Internet Protocol (IP) effort has emerged as a natural follow-on to the development of the Arpanet; these protocols are now being used to provide communication among resources on the Arpanet, the Bay Area Packet Radio Network, and other systems [Postel, 1979, 1980].

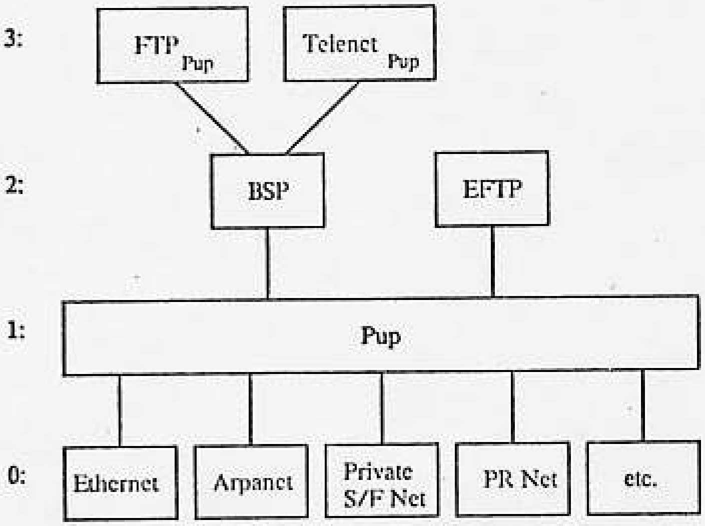

Figure 4.1: Part of the Xerox Pup internetwork architecture.

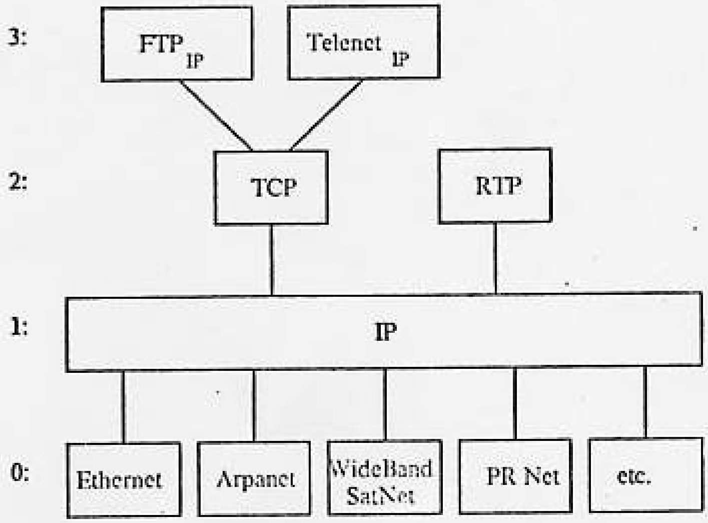

Figure 4.2: Part of the Arpa IP internetwork architecture.

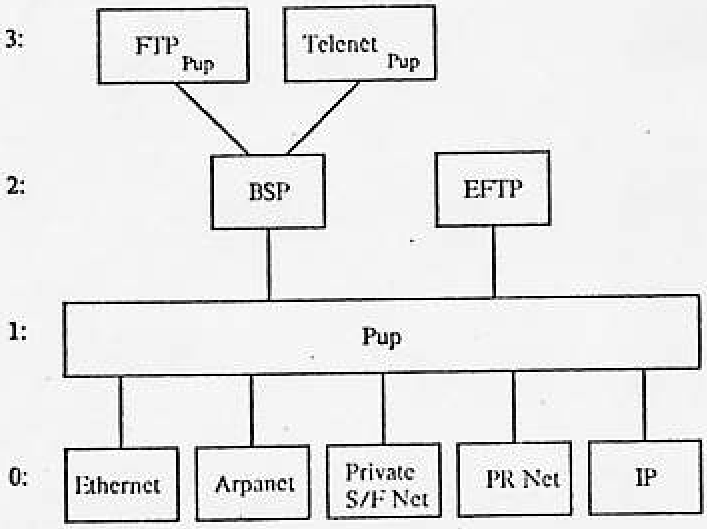

Figure 5.1: The Xerox Pup internetwork architecture, with the IP internet as one packet transport mechanism.

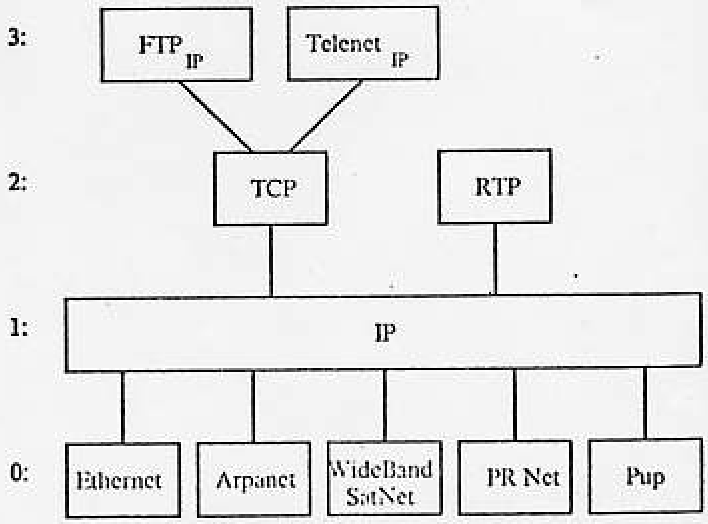

Figure 5.2: The Arpa IP internetwork architecture, with the Pup internet as one packet transport mechanism.

Figure 4.2 shows a portion of the IP architecture. Again, at the bottom (level 0) are several different networks used to transport packets. Level 1 defines the internetwork datagram as part of the Internet Protocol (IP). Above that layer we find the Transmission Control Protocol (TCP), which provides an error-controlled, flow-controlled stream of bytes. At level 3, we find several higher-level protocols, including the File Transfer Protocol (FTP), and the Telnet protocol.

The key to both designs is the uniformity at level 1 -- a standard internetwork packet format, which can be encapsulated for transmission through individual networks; the Pup level 1, however, is different from the IP level 1.

Note also that both designs include higher-level internetwork protocols known as FTP and Telnet; while the names are the same, these do not represent the same protocol, and are not compatible. Once again, the purpose of the present work is to provide mutual support of different internetwork protocols, but we are not trying to directly bridge the gap between these divergent higher level protocols. Where there might be cause for confusion we usually refer to these different designs as FTP$_{Pup}$ and Telnet$_{Pup}$, as distinct from FTP$_{IP}$ and Telnet$_{IP}$.

The evolution of these two systems has, in part, kindled our interest in mutual encapsulation. It has been proposed, for example, that the Pup internet could be used as a backup communications path if portions of the IP internet were unavailable. Furthermore, there are several institutions which are already part of the Arpa IP community, but which will be acquiring some specialized systems which make use of the Pup protocols. Within such a site it will be simple to use either the IP protocols or the Pup protocols, as dictated by particular applications. But how are we to provide long-haul communication to other sites -- particularly other sites running Pup-based software? One approach would be to extend the Pup internetwork directly to all of these sites; but that would be a substantial effort.

Most of these Pup-based sites, however, already have connections to the IP environment. Therefore, it now seems attractive to integrate the entire IP internet as one transport mechanism available for Pup-based communication (see Figure 5.1); this is the primary pragmatic objective which originally motivated this work, and is the basis for current experimentation. Conversely, however, it might also be possible to use the Pup internet as an alternative path for IP traffic (see Figure 5.2). Figure 5.3 shows the general way in which mutual encapsulation could be used between the Pup and IP worlds.

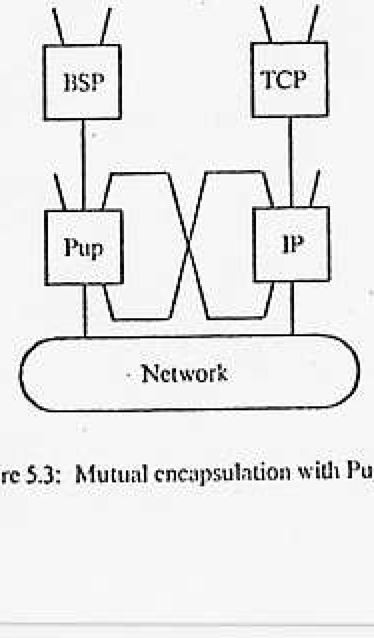

Figure 5.3: Mutual encapsulation with Pup & IP.

3. A note on notation

In typical network architectures, such as Figures 5.1 and 5.2, it's important to note that each line between two layers is a bi-directional path: an outbound datagram can be encapsulated and handed down to a network, while an inbound packet is decapsulated and handed up from the network. At any layer it's always evident how encapsulation and decapsulation are performed: you encapsulate with a header as you go "down" and decapsulate as you come "up."

As shown in Figure 5.3, however, mutual encapsulation can introduce non-hierarchical loops in the figure, distorting the notion of "up" and "down." When a packet is passed from the box called Pup to the box called IP, does this mean that we decapsulate a Pup to find an IP inside, or are we supposed to encapsulate the Pup inside an IP?!

The strictly linear, vertical format provided some important extra information in making this decision, which is no longer appropriate. The situation is aggravated as we try to extract individual paths from Figure 5.3; Figure 6.1 shows two different configurations in a horizontal format, which is now ambiguous as we move from "left" to "right." As we saw, the encapsulation/decapsulation operation is bi-directional, but asymmetric; and we've lost the hint about which way is up!

To solve this notational problem we have adopted a different form, which re-introduces the needed information (see Figure 6.2). Each transformation is now indicated by a small box inside of a larger one, suggestive of the encapsulation process. If something comes in the small box it is encapsulated before coming out of the big box; if something enters the big box, it is decapsulated before passed out through the small box. Thus, the top portion of Figure 6.2 shows some data coming in from the left, encapsulated in a Pup, which is then encapsulated in an IP packet.

Each connection to a box is labelled with the type of packet it can produce or receive at that port. Thus, these figures can be manipulated like dominos, lining up matching packet types to produce a series of transformations.

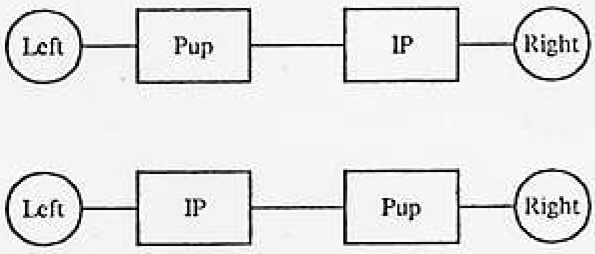

Figure 6.1: Two ambiguous representations of encapsulation.

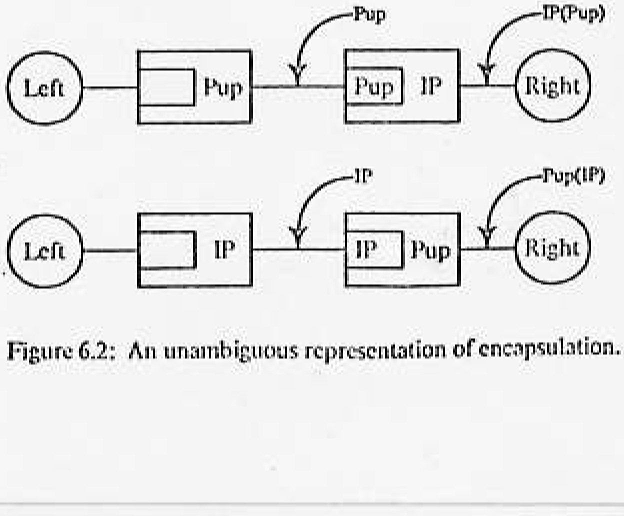

Figure 6.2: An unambiguous representation of encapsulation.

Figure 6.2 also shows a functional notation that can be very useful in describing encapsulation and decapsulation: IP(Pup) is an IP holding a Pup. If we encapsulate this packet inside an Arpanet packet, this produces A(IP(Pup)); on an Ethernet system it would become E(IP(Pup)). Similarly, if this Pup is carrying a BSP packet which in turn is part of a file transfer, the packet would be represented as Pup(BSP(FTP)).

The functional notation can be extended to record additional information, particularly the destination address used in each level of encapsulation. This is especially important in working out very detailed examples, as one must determine which layers must be able to do appropriate address mapping. In the example above the Pup may have a destination drawn from the Pup address space (<PupNetX,HostY>), while the IP packet has a destination drawn from the IP address space (<IPNetA,HostB>); this packet then becomes:

$$IP_{\langle IPNetA,HostB\rangle}(Pup_{\langle PupNetX,HostY\rangle}).$$

When encapsulated for transmission through a particular Ethernet network, the Ethernet header contains only a destination host address, and the packet becomes:

$$E_{LocalHost\#}(IP_{\langle IPNetA,HostB\rangle}(Pup_{\langle PupNetX,HostY\rangle})).$$

4. Ways to provide internetwork communication, with and without mutual encapsulation

Against this background we can now elaborate on the basic problem: we have two similar internet designs (Pup and IP) and a set of individual networks which can be physically interconnected. What remains is the design of suitable software in the hosts and gateways to support the various forms of internet communication, including mutual encapsulation. There are four cases of interest:

1. Pure Pup communication.
2. Pure IP communication.
3. Pup communication, carried by IP.
4. IP communication, carried by Pup.

It is evident that cases 1 and 2 are very similar, and cases 3 and 4 are the dual of each other. The latter two do involve an additional level of encapsulation; as we shall see, this may take place either in the hosts, or in an intermediate gateway.

To make things more concrete, let's consider the following example (as shown in Figure 7):

1. There are two physical sites involved, named West and East.
2. Each of the sites is served by a local computer network -- in this case an Ethernet system.
3. Each site has a number of computers, using various operating systems and running many different applications programs. Each of these machines has a host number on its local network (shown here as 1, 2, and 3 at both sites).
4. Some of these programs use the Pup protocols, while others use the IP protocols.
5. There is a long-haul store-and-forward network which can be used between the two sites -- in this case the Arpanet.
6. There are hosts at each site which can be connected to both the local Ethernet installations and the Arpanet; thus, they can serve as internetwork gateways (G_{west} and G_{east}). Each of these hosts has an Arpanet address (shown here as 61 and 62).

The objective, of course, is to explore the kinds of communications available, and determine the software needed in the gateways and (possibly) in the hosts.

4.1. *Pure Pup communication*

As we have seen, there may be some hosts and servers at each site which only speak the Pup protocols. Within the site, these hosts can communicate directly; for a user at one site to access a server at the other site, however, will require a Pup-based gateway.

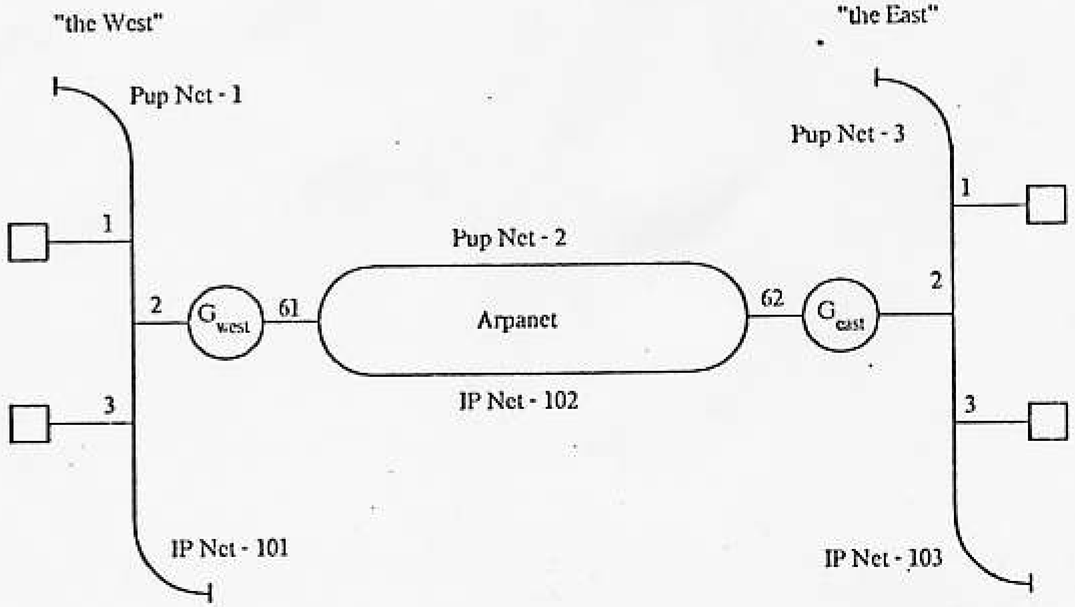

Figure 7: A concrete example, with 3 networks.

Network addresses in the Pup design take the form of a triple: <network number, host number, socket number>; since the socket number is not needed in most of this discussion, it will usually be left out. Thus, each of the three networks would be part of the Pup internet, and each net is given a network number drawn from the Pup network address space (in this case, they are called PupNet-1, PupNet-2, and PupNet-3).

To send an FTP packet from Host1 in the West to Host1 in the East, the following steps take place:

1. Host1, in the West, generates a Pup of the form Pup(BSP(FTP)); its internet source is <PupNet-1,Host1> and its destination is <PupNet-3,Host1>.
2. The Pup software in Host1 consults its routing table and finds that the best path to PupNet-3 is through a gateway on its local net: gateway G_{west}, known locally as the destination <PupNet-1, Host2>.
3. The Pup is now encapsulated for transmission through the local Ethernet link, and becomes E(Pup(BSP(FTP))); the Ethernet encapsulation indicates that the destination is Host2, and that this Ethernet packet contains a Pup.
4. At the gateway, Host2, the Ethernet driver receives the packet, and recognizes that it is a Pup. It is decapsulated, to become Pup(BSP(FTP)), and is handed to the Pup software in the gateway.
5. The routing software recognizes that the best path to the Pup destination <PupNet-3, Host1> is through G_{east} -- Host62 on the Arpanet. The packet is encapsulated for transmission through the Arpanet, and becomes A(Pup(BSP(FTP))); the Arpanet encapsulation indicates that the destination is Host62, and that this Arpanet packet contains a Pup.
6. At gateway G_{east} (Arpanet address Host62) the Arpanet driver receives the packet, and recognizes that it is a Pup. It is decapsulated, to become Pup(BSP(FTP)), and is handed to the Pup software in the gateway.
7. The Pup software recognizes that it is directly connected to the destination network, <PupNet-3>. The packet is again encapsulated for transmission through the local Ethernet system to Host1, and becomes E(Pup(BSP(FTP))).
8. The packet finally reaches its destination, Host1 on PupNet-3. Once again the packet is decapsulated to produce Pup(BSP(FTP)), and this is handed on to the appropriate process in that host.

Note that all of this communication was based upon Pups, using the local Ethernet systems and the Arpanet as independent Pup transport networks. All of the network and host naming conventions are entirely within the purview of the Pup internet environment. Most importantly, each "pure Pup" gateway merely had to perform this transformation (see Figure 8.1):

E(Pup(BSP(FTP))) <=> Pup(BSP(FTP))
<=> A(Pup(BSP(FTP))).

4.2. Pure IP communication

In addition to Pup software, however, each site also has machines running IP software. Within the site, these hosts can also communicate directly; but inter-site communication will require an IP-based gateway.

The IP protocols also use a hierarchical address; thus, in the IP world, our three networks must be assigned network numbers from the IP network address space (in this case, IPNet-101, IPNet-102, and IPNet-103).

The basic operation of this approach is similar to case 1 above, but using different forms of encapsulation; the details will not be repeated here. In this case (see Figure 8.2) each "pure IP" gateway performs this transformation:

E(IP(TCP(FTP))) <=> IP(TCP(FTP))
<=> A(IP(TCP(FTP))).

4.3. Pup communication, carried by IP

This third alternative is the more interesting case: Pup-based communication, using the whole IP system as one packet transport mechanism. For this to work, we must encapsulate Pup packets inside of IP packets; this is an extra layer of encapsulation not present in the first two examples.

This IP encapsulation can take place in one of two places: either in the end hosts, or in an intermediate gateway. If the encapsulation is done in the hosts there will be a need for special software there, but the system can use a regular IP gateway. Conversely, encapsulation in a gateway means that the host Pup software can go unchanged, but the gateway must be modified. (Note that in this discussion we have shown only a single network -- the Arpanet -- as part of the IP system; in general, the IP internet could be spanning multiple networks.)

4.3.1. Additional encapsulation in the hosts

In this case, we are trying to do a file transfer between two Pup hosts: the source host in the West, however, will initially encapsulate each Pup inside an IP packet. This IP can then be carried through the local network and on to a "pure IP" gateway. These are the steps that would be taken:

1. Host1, in the West, generates a Pup of the form Pup(BSP(FTP)); its internet source is <PupNet-1,Host1> and its destination is <PupNet-3,Host1>.
2. The Pup software in Host1 consults its routing table and finds that the best path is through the IP; the packet is handed to an "IP driver" in Host1.
3. The IP module in Host1 encapsulates the Pup inside an IP, producing a packet of the form IP(Pup(BSP(FTP))). Note that the original Pup was destined for the Pup address <PupNet-3,Host1>; when encapsulated, the IP driver must be able to correctly map this Pup address into an IP address: <IPNet-103,Host1>.
4. The IP module in Host1 consults its IP routing table, and finds that the best path to IPNet-103 is through a gateway on its local net: gateway G_{west}, known locally as the destination <IPNet-101, Host2>.
5. The IP is now encapsulated for transmission through the local Ethernet link, and becomes E(IP(Pup(BSP(FTP)))); the Ethernet encapsulation indicates that the destination is Host2, and that this Ethernet packet contains an IP.
6. At the gateway, Host2, the Ethernet driver receives the packet, and recognizes that it is an IP. It is decapsulated, to become IP(Pup(BSP(FTP))), and is handed to the IP software in the gateway.
7. This IP now continues through the IP internet -- through the Arpanet and gateway G_{east} until it reaches the destination host, <IPNet-103,Host1>.
8. At the destination host the packet is decapsulated in the Ethernet driver, to produce IP(Pup(BSP(FTP))). Recognized as an IP packet, it is handed to the IP software.
9. The IP header indicates which process is to receive the packet; in this case it is the Pup software.
10. The Pup software finally looks at the real Pup destination and the packet is handed on to the appropriate process in that host.

Once again, note that the entire IP system is treated as one network by Pup; any Pup-based machine doing host encapsulation is viewed as being directly connected to this one large network. Since the encapsulation of Pups inside IP packets takes place in the hosts, this approach can make use of unmodified, "pure IP" gateways. In this example, the local networks in the West and the East can really be thought of as IP-based systems, with each network normally assigned an IP network number by the administrators of the IP internet. For hosts using Pup communication, however, the whole IP network is given one network number drawn from the Pup network address space, and each machine doing host encapsulation is given a Pup host number on that network.

As occurs with the addition of any new network, the existing Pup-based host software must be modified to incorporate the IP driver -- the IP(Pup) module. Note that this operation requires more than just reformatting a header; this host software must be able to correctly map Pup addresses into the corresponding IP addresses (e.g., it must know that the destination Pup address <PupNet-3,Host1> corresponds to the IP address <IPNet-103,Host1>). Since both of these name spaces are subject to change by their respective groups, the host software module must be prepared to accommodate such changes, and manage its internal tables in the face of dynamically changing internet configurations.

4.3.2. Additional encapsulation in the intermediate gateways

Instead of doing the extra IP encapsulation in the host, we could choose to put this function in an intermediate gateway. Source hosts could run un-modified Pup software, sending Pups to the nearest gateway: the modified gateway would then encapsulate them inside IP packets for further delivery. These are the steps that would be taken:

1. Host1, in the West, generates a Pup of the form Pup(BSP(FTP)): its internet source is <PupNet-1,Host1> and its destination is <PupNet-3,Host1>.
2. The Pup software in Host1 consults its routing table and finds that the best path to PupNet-3 is through a gateway on its local net: gateway G_{west}, known locally as the destination <PupNet-1,Host2>.
3. The Pup is now encapsulated for transmission through the local Ethernet link, and becomes E(Pup(BSP(FTP))); the Ethernet encapsulation indicates that the destination is Host2, and that this Ethernet packet contains a Pup.
4. At the gateway, Host2, the Ethernet driver receives the packet, and recognizes that it is a Pup. It is decapsulated, to become Pup(BSP(FTP)), and is handed to the Pup software in the gateway.

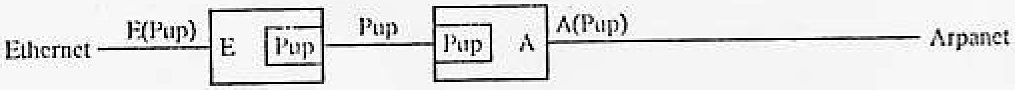

Figure 8.1: Pure Pup Gateway.

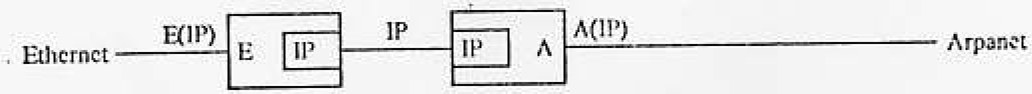

Figure 8.2: Pure IP Gateway.

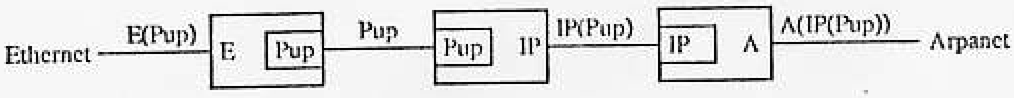

Figure 8.3: IP (Pup) Gateway.

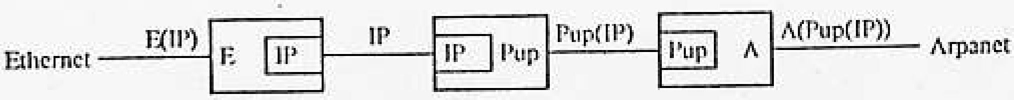

Figure 8.4: Pup (IP) Gateway.

5. The Pup routing software recognizes that the best path to the Pup destination <PupNet-3,Host1> is to gateway G_{east} through the IP internet. The packet is encapsulated for transmission through the IP internet, and becomes IP(Pup(BSP(FTP))): the IP encapsulation indicates that the destination is the Pup module at host <IPNet-102,Host62>.
6. The IP software now consults its routing table, and finds that the best route to <IPNet-102,Host62> is through the Arpanet: the packet is handed to the Arpanet driver, and becomes A(IP(Pup(BSP(FTP)))), addressed to Host62 on the Arpanet.
7. At Arpanet address Host62 the Arpanet driver receives the packet, and recognizes that it is an IP. It is decapsulated, to become IP(Pup(BSP(FTP))), and is handed to the IP software in the gateway.
8. The IP software now recognizes that the destination IP address is the Pup process in this host: at this point the extra layer of encapsulation is undone, and the packet again becomes Pup(BSP(FTP)), and is handed to the Pup module in G_{east}.
9. The Pup software recognizes that it is directly connected to the destination network, <PupNet-3>. The packet is again encapsulated for transmission through the local Ethernet system to Host1, and becomes E(Pup(BSP(FTP))).
10. The packet finally reaches its destination, Host1 on PupNet-3 Once again the packet is decapsulated to produce Pup(BSP(FTP)), and this is handed on to the appropriate process in that host.

Thus, the systems running at the source and destination machines use unmodified Pup software. The gateways, however, must be modified to include a new IP driver at level 0 that will perform the extra encapsulation -- they are "IP(Pup)" gateways performing this transformation (see Figure 8.3):

E(Pup(BSP(FTP))) <=> Pup(BSP(FTP))
<=> IP(Pup(BSP(FTP)))
<=> A(IP(Pup(BSP(FTP)))).

The entire IP system is treated as one Pup network, with a network number drawn from the Pup address space; each of the special Pup gateways which connects to the IP system must have a Pup host number to identify it on that network. The Pup gateways will use these addresses when exchanging routing tables through the IP system, for example.

The local networks, however, are treated as Pup-based systems, and do not need a network number from the IP address space.

4.4. IP communication, carried by Pup

The third case above described ways to use Pup communication, built on top of the IP system. This fourth case is the dual of that: using the IP protocols, built on top of the Pup system. The fundamental operation here is the same as described before, although the details of encapsulation will vary. Again, this encapsulation of IP packets inside of Pups could take place in the hosts, or in the gateways.

4.4.1. Additional encapsulation in the hosts

If the encapsulation is done in the hosts the IP software there must be modified to include a Pup driver, but the existing Pup gateways can be used without change.

4.4.2. Additional encapsulation in the intermediate gateways

If the encapsulation is done in the gateways they must be modified, but the IP software in source and destination hosts can be used without change.

In this case, the gateways become "Pup(IP)" gateways, performing this transformation (see Figure 8.4):

E(IP(TCP(FTP))) <=> IP(TCP(FTP))
<=> Pup(IP(TCP(FTP)))
<=> A(Pup(IP(TCP(FTP)))).

5. Selecting an overall strategy

We've now seen six alternative communications paths (pure Pup, pure IP, and four with additional encapsulation): these have also made use of four possible gateway designs (pure Pup, pure IP, IP(Pup), and Pup(IP)).

How can we make some sense out of this situation? If we had all Pup or all IP traffic the choice would be clear; but remember that we are trying to support both Pup and IP protocols, while sharing some communications facilities -- that was the purpose of mutual encapsulation.

Again, we will examine the case in which -- where needed -- Pup traffic is carried inside of IP packets. (The complementary case in which IP packets are carried inside of Pups is not fundamentally different.) Given this assumption, carrying regular IP traffic is now straightforward: it uses regular IP software at the hosts, and regular "pure IP" gateways.

So the only significant choice to be made is the method of transporting Pups -- using host encapsulation or gateway encapsulation. As we've seen, host encapsulation would be able to take advantage of unmodified IP gateways, but would necessitate modifications to all existing Pup systems: in some applications that are treated as "turn-key" systems, that might not even be feasible. Instead of changing all of the user hosts, however, we can encapsulate into IP packets at the gateways. This leaves the Pup systems untouched, but does require construction of a new, dual-purpose gateway: it is a combined "pure IP" gateway and "IP(Pup)" gateway, as shown in Figure 9.1.

The most important result of this design is that it can accept unmodified IP or Pup packets (at the left of Figure 9.1), and forward them through a system that only accepts IP packets (at the right).

As we indicated earlier, one of the immediate motivations for this work was the need to support Pup-based communication to sites which were not currently part of the Pup internet but were part of the IP world. This final strategy is the one that has been adopted: for sites that only have an IP connection, Pups can be encapsulated in the gateways for long-haul transmission. This does require the construction of a modified gateway program for access to a Pup-based network, but requires no changes to any existing user programs or servers.

This does mean, however, that any regular site on the Pup internet must have a gateway with an "IP driver" in order to send Pups to one of the remote locations. Alternatively, one could even implement both host encapsulation of Pups and gateway encapsulation, and the two can communicate with each other: a source host could perform the encapsulation, while the decapsulation is performed in the destination gateway.

Note also that if we had chosen to embed IP packets inside of Pups, the dual-purpose gateway would combine a "pure Pup" gateway and a "Pup(IP)" gateway, as shows in Figure 9.2.

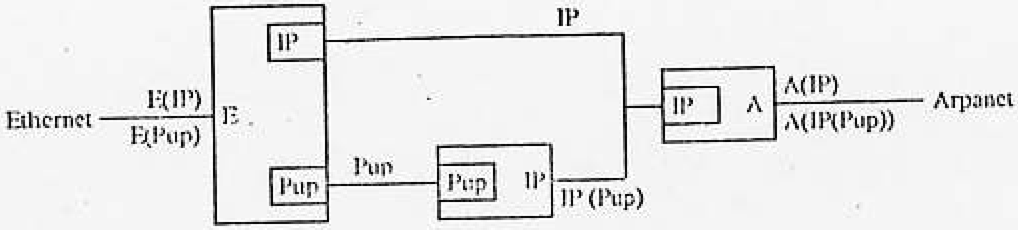

Figure 9.1: IP only on long haul; IP & IP (Pup) Gateway.

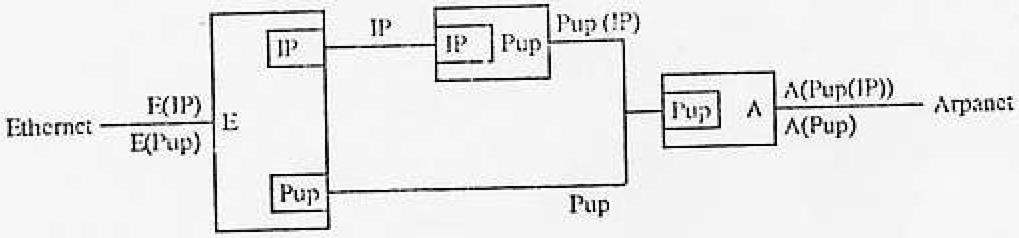

Figure 9.2: Pup only on long haul; Pup & Pup (IP) Gateway.

6. Mapping of internetwork addresses

Astute readers may have noticed an important issue which has not yet been treated in detail: the *mapping* of internetwork host addresses into actual physical host addresses, also known as *address translation*. This problem occurs in the existing internetwork architectures, and emerges again when one considers the use of mutual encapsulation.

Recall that both the Pup and IP designs make use of a hierarchical address space to access all internet hosts [Shoch, 1978]: since the host number is defined only in the context of a particular network, we sometimes refer to this as *network-specific* or *network-relative* addressing. In the Pup system, each host is identified with a pair <Network#,Host#>. Eight bits are used for the network number, allowing addressing of 256 total networks. Pup host addresses are also eight bits wide, supporting up to 256 hosts per network.

In the IP system there is also an 8-bit network field, combined with a 24-bit field containing the "rest" of the address; these 24 bits can be used in any network defined manner, and may support additional levels of the addressing hierarchy.

In both of these designs one could provide mechanisms to support extended addressing, perhaps with a distinguished address used as an escape indicator.

One key issue on any network is the manner in which we map from the internet host address to the actual network address -- particularly as a packet is encapsulated at a gateway for transmission to a particular host within a net. In general, this may require some form of table look up. In most networks, however, there is a straightforward degenerate case: on Ethernet systems and on the Arpanet, for example, the internet host address is the same as the regular network address.

Unfortunately, there is at least one circumstance in which this mapping will not be so simple: when the network's address space is larger than the internet host address space. If there are more actual hosts on one network then there are internet addresses, then it may not be possible to address them all. In the lifetime of the Pup design, though, we do not expect to see very many systems that are expected to support more than 256 Pup-based hosts on one network; in any case, it is usually quite simple to divide the network into two systems, or perhaps use two network numbers for different parts of the same system.

More pressing, however, is the case where one wants to reach a limited subset of the hosts attached to a large network; then, it may be necessary to map the internet identifier in some manner. Furthermore, there are systems which support a limited number of hosts, but where the format used for the network address field is very wide -- wider than the internet address can directly represent. Even with very few hosts attached, it may be necessary to map internet addresses into the local address space.

This has already been required on some existing networks (such as the Packet Radio Network): it emerges dramatically when one tries to use the entire IP system as a single packet transport network.

6.1. *Mapping Pup internet addresses on the Packet Radio Network*

The Bay Area Packet Radio Network (the PRNet) has been integrated as one packet transport mechanism in the Pup environment, carrying Pup internet packets between two buildings in the San Francisco Bay Area [Kahn, *et al.*, 1978; Shoch & Stewart, 1979]. In the Pup world the PRNet has been given a Pup network number, and each Pup-based host must be assigned an eight bit host number.

Unfortunately, the PRNet uses a structured address field which is 16 bits wide, while the Pup host field is 8 bits wide. It is very unlikely that any single PRNet would ever have more than 256 Pup-based hosts; but it is necessary to map the 8-bit internet host address into the 16 bit PRNet address. The mapping operation takes place in the network-specific driver for any Pup-based host connected to the PRNet: there is a simple table there which can be used to map internet addresses into PRNet addresses. There are some procedures required to maintain and update this table; fortunately, this is entirely confined to the set of hosts on the PRNet.

In the near term one could solve this specific problem by expanding the size of the Pup host address field. But that only buys a little time, and can not solve some of the more general cases described below.

6.2. *Mapping Pup internet addresses on the IP internet.*

As we've seen, mutual encapsulation provides a means to carry Pup packets through the IP internet -- the entire IP internet can be treated as a single Pup packet transport mechanism. Thus, the IP system will be given a Pup network number, and each Pup-based destination within the IP world must then also be given a Pup host address on this network.

The boundary of the IP network is determined by the point at which the IP(Pup) encapsulation will take place -- either right in the host (with host encapsulation) or at a gateway (with gateway encapsulation). In either case, this is the point at which a Pup is handed to the IP driver. But the Pup only has an 8-bit host address inside, and the IP driver needs to send its packet to a regular IP address: that destination may be many networks away within the IP system, and requires a full 32-bit IP internet address.

This initial address mapping then determines the point at which the decapsulation must take place: the IP containing a Pup may be sent to the destination gateway (for gateway decapsulation) or to the destination IP host (for host decapsulation).

Initially, strategies for maintaining these address mappings can be similar to the Pup-to-PRNet translation -- use a small table in the IP driver.

Again, the need to map addresses is the most general procedure required for internetworking, and cannot be avoided. In the near term, of course, the ability to equate internet host addresses and physical host addresses provides a simple approach, and limited table lookup will certainly solve some special cases. In the long run, however, we must be prepared to move beyond simple table lookup, and develop dynamic procedures for mapping both names and addresses in a large internetwork system.

7. Some final words

We have described at some length the way in which mutual encapsulation can be used to support both Pup and IP communication. This extended example has helped to demonstrate some of the procedures that will be used in an actual implementation. These techniques can be used to extend the range of each internet, or to provide a form of mutual backup in case either system experiences a partial failure.

It's important to recognize, however, that the basic ideas transcend the particular example: mutual encapsulation of protocols is a technique that can find applicability in a wide variety of situations, where it can be used to extend the range of any particular pair of protocols. The examples above have also helped to highlight some of the important issues involved: where will the extra layer of encapsulation be implemented, what information will be needed to perform the encapsulation, how must addresses be mapped, and what other parts of the system must be changed.

8. Acknowledgements

The idea of mutual encapsulation has been explored at various times in the development of internetwork protocols; the work described here was triggered by a desire to solve the combined Pup/IP communication problem. Early discussions took place in the summer of 1979, and Danny Cohen and Bob Sproull developed an initial proposal for attacking the problem. Much of the detailed design and implementation planning took place at a meeting held at Xerox Parc in late 1979 [Cohen, 1979]; we very much appreciate the energy contributed by the participants in that meeting. In addition, valuable comments on this paper were received from Yogen Dalal, Larry Stewart, and Mary Artibee.

Finally, we believe that the term "encapsulation" was first used during the design of Pup, and the phrase "mutual encapsulation" was originally coined by Bob Metcalfe, in a slightly different context. In the early days at Parc we discussed ways to encapsulate Pups inside Arpanet packets. At the same time, we had an Arpanet host that did not have a direct Imp interface: regular Arpanet NCP packets were to be encapsulated inside of Pups, for transmission to a front-end that really had the Imp hardware. Each protocol did double duty, encapsulating the other when necessary -- and thus emerged the term mutual encapsulation.

9. Bibliography

[Boggs, *et al.*, 1980]
D. R. Boggs, J. F. Shoch, E. A. Taft, R. M. Metcalfe, "PUP: An Internetwork Architecture," *IEEE Transactions on Communications*, April 1980.

[Cerf & Kirstein, 1978]
V. G. Cerf and P. T. Kirstein, "Issues in packet-network interconnection," *Proceedings of the IEEE*, 66:11, November 1978, pp. 1386-1408.

[Cohen, 1979]
D. Cohen, *Summary of the Arpa/Ethernet community meeting*, Internet Experiment Note #126, November 1979.

[Kahn, *et al.*, 1978]
R. E. Kahn, S. A. Gronemeyer, J. Burchfiel, and R. C. Kunzelman, "Advances in packet radio technology," *Proc. of the IEEE*, 66:11, November 1978, pp. 1468-1496.

[Metcalfe & Boggs, 1976]
R. M. Metcalfe and D. R. Boggs, "Ethernet: Distributed packet switching for local computer networks," *Communications of the ACM*, 19:7, July 1976, pp. 395-404.

[Postel, 1979]
J. Postel, ed., *DOD Standard Internet Protocol*, Internet Experiment Note #123, USC/ISI, December 1979.

[Postel, 1980]
J. Postel, "Internetwork protocols," *IEEE Transactions on Communications*, April, 1980.

[Shoch, 1978]
J. F. Shoch, "Inter-network naming, addressing, and routing," *Proc. of the 17th IEEE Comp. Soc. Int. Conf. (Compcon Fall '78)*, Washington DC, September 1978, pp. 72-79.

[Shoch, 1980]
J. F. Shoch, "The PUP Internet: Protocols, Servers, and Applications," *to appear*, 1980.

[Shoch & Hupp, 1979]
J. F. Shoch and J. A. Hupp, "Performance of an Ethernet local network -- a preliminary report," *Proc. of the Local Area Communications Network Symposium*, MITRE/NBS, Boston, May 1979, pp. 113-125. Revised version published in *Proc. of the 20th IEEE Comp. Soc. Int. Conf. (Compcon '80 Spring)*, San Francisco, February 1980, pp. 318-322.

[Shoch & Stewart, 1979]
J. F. Shoch and L. Stewart, "Interconnecting local networks via the Packet Radio Network," *6th Data Communications Symposium*, Pacific Grove, November 1979, pp. 153-158.

[Sunshine, 1977]
C. Sunshine, "Interconnection of computer networks," *Computer Networks*, 1, 1977.