The Pyramid IP to X.25 Protocol Interface: Merging DDN and PDN Approaches

Ramón Cáceres

Pyramid Technology Corporation
1295 Charleston Road
Mountain View, California 94039

ABSTRACT

The number of different networks now accessible with the DARPA family of protocols is considerably larger than the traditional set of Ethernet and ARPANET. Of these network types, one that promises to become increasingly important is X.25 networks. More specifically, connecting the DARPA family of protocols to X.25 networks involves creating an interface between the IP and the X.25 protocols. Pyramid Technology Corporation recently implemented this interface for its *dualPort* OSx[1] operating system, extending its DARPA-style networking capabilities to the X.25 portion of the Defense Data Network (DDN) as well as to a number of X.25 Public Data Networks (PDN). This paper covers some of the more interesting design and implementation issues encountered while developing an IP to X.25 interface that complies with both DDN and PDN requirements.

Introduction

The family of standard computer communication protocols adopted by the Defense Advanced Research Projects Agency (DARPA) has found wide use in providing host-to-host communication over a large variety of computer networks. These networks are in turn based on a number of different lower level protocols, such as the Ethernet 1.0 protocols in the case of the Ethernet local area network, and the 1822 protocol in the case of the ARPANET wide area network.

One very important class of these networks is X.25 networks, based on X.25 and its related standard protocols put forward by the Consultative Committee for International Telegraph and Telephone (CCITT). X.25 networks include the X.25 portion of the Defense Data Network (DDN), as well as a number of X.25 Public Data Networks (PDNs).

Using the DARPA family of protocols over X.25 networks involves creating an interface between the DARPA IP protocol and the CCITT X.25 protocol, as shown in Figure 1. Pyramid Technology Corporation recently implemented this interface for its *dualPort* OSx operating system, a multi-environment port of the AT&T System V and the University of California 4th Berkeley Software Distribution (BSD) versions of the UNIX[2] operating system. This extends OSx's DARPA-style networking capabilities to the X.25 portion of the DDN as well as to a number of X.25 PDNs.

This paper describes some of the more interesting design and implementation issues dealt with at Pyramid while developing an IP to X.25 interface, specially one that complies with both

[1] *dualPort* and OSx are trademarks of Pyramid Technology Corporation
[2] UNIX is a trademark of AT&T

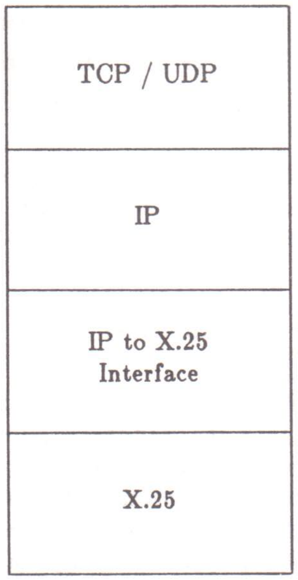

Figure 1 - IP to X.25 Protocol Layering

DDN and PDN requirements. Section 1 will give a short overview of the DARPA family of protocols, emphasizing IP, while Section 2 will provide an introduction to the CCITT set of X.25-related protocols, and X.25 in particular. Section 3 will describe some of the generic design issues involved in creating an IP to X.25 interface. Following, Section 4 will compare and contrast the DDN and PDN approaches to interfacing IP to X.25, as defined by two different standards. Section 5 will then describe the steps taken at Pyramid to merge these two approaches in a single interface, and Section 6 will go into some of the more interesting implementation details encountered while developing the Pyramid IP to X.25 interface. Finally, section 7 will outline some of the remaining work in this area.

1. The IP Protocol

The DARPA family of standard protocols includes a virtual terminal protocol (TELNET), the File Transfer Protocol (FTP), the Simple Mail Transfer Protocol (SMTP), the Transmission Control Protocol (TCP), and the Internet Protocol (IP) [4]. Together they provide a comprehensive set of network services over both local area and wide area networks. As widely accepted and machine-independent standards, they can be used for communication among a large variety of computer equipment from many different vendors.

IP, in particular, adds some very important functionality to the DARPA protocol family. As its name implies, IP handles addressing and routing between separate logical and/or physical networks, i.e., an internetwork or internet.

Crucial in this internet role is the IP addressing scheme. IP addresses consist of 32 bits typically divided into two parts: the network address and the host address. Together, the two provide the IP layer of the hosts in an internet with all the information necessary for routing IP packets from their origin to their destination.

Also included in the IP specification is provision for networks whose maximum packet size is smaller than the maximum IP packet size. Through the notion of a maximum transmission unit, IP packages data from higher level protocols into IP packets of a size known to be acceptable to lower level protocols and physical networks. This is commonly referred to as IP fragmentation.

IP is a datagram protocol, which means that every unit of data handled by the protocol is treated individually from all others. This is in contrast to stream or virtual circuit protocols, which typically maintain a serial relationship between subsequent units of data for the same physical or logical destination, thereby insuring both reliable and properly ordered delivery. Implied in the concept of datagrams is the fact that each datagram must by necessity contain enough information to traverse a path from its origin to its final destination. In addition, datagram protocols do not usually insure reliable or properly ordered data delivery. Naturally, data is not purposefully lost without discretion, but rather when conditions necessitate it such as upon heavy network congestion. However, it is important to note that IP will allow data to be dropped and delivered out of order.

2. The X.25 Protocol

The CCITT X.25 and its related protocols provide a similar set of network services to the DARPA protocols. They too can be used for communication among a large variety of computer equipment from many different vendors. These protocols include X.25, X.28, X.29, and X.3, among many others [7].

X.25 provides functionality that includes aspects of both TCP and IP. It is a reliable virtual circuit protocol covering the physical, frame, and packet level of the International Standards Organization (ISO) Open Systems Interconnect (OSI) model. In X.25 terminology, these levels are referred to as X.25 Level 1, X.25 Level 2, and X.25 Level 3, respectively.

Similar to IP, X.25 is a host to host protocol. As such, X.25 maintains a host addressing scheme obeying the CCITT X.121 standard. X.25 addresses are variable length, encoded two digits per byte as Binary Coded Decimals (BCD). An X.25 address is limited to a maximum of 14 BCD digits. This limit is enforced by a 4-bit address length field in packets which explicitly contain X.25 addresses. X.25 addresses are broken up into a 4-BCD-digit Data Network Identification Code (DNIC), and a variable length host address within that data network, sometimes referred to as the National Terminal Number (NTN).

In contrast to IP, X.25 is a virtual circuit protocol. An important fact to keep in mind regarding virtual circuit protocols is that there is a very definite and unvoidable overhead incurred before data can be transferred to a destination for which a virtual circuit does not already exist. In short, a virtual circuit must be explicitly established between the source and destination entities before any higher level data can flow. For establishment to take place, virtual circuit protocols usually go through an exchange of control information that constitutes what is usually termed a handshake.

In X.25, the sequence of steps necessary to establish a virtual circuit include the exchange of a call request packet, and either a call accept packet if the called party is willing to go ahead with the transaction, or a clear request packet if the called party refuses to go ahead with the transaction. Until a call request-call accept handshake is successfully performed, no higher level data transfer can take place over an X.25 virtual circuit. In X.25 terminology, this process is termed establishing a call.

3. General Design Issues in Interfacing IP to X.25

Many issues need to be resolved when the IP and X.25 protocols are interfaced. A lot of these can be considered purely implementation decisions, but several stand out as more general design problems: IP to X.25 address translation, X.25 virtual circuit management, the ability to perform packet fragmentation and reassembly, and internetwork routing functionality.

3.1. IP to X.25 Address Translation

Both IP and X.25 maintain their own host to host addressing schemes, as described earlier. On outgoing IP packets, the interface must translate IP addresses into X.25 addresses. Similarly, on incoming X.25 call requests, the interface must translate X.25 addresses into IP addresses. Several ways to perform this two-way translation come to mind.

The most straight forward way is for the interface to keep a table of IP to X.25 address mappings. This is very easy to build and maintain in a general sense, but presents some practical drawbacks. In current UNIX systems, the IP protocol is typically in the kernel, while the X.25 protocol is found either in a hardware controller or also in the kernel. Thus, the most logical implementation of the IP to X.25 interface is a piece of kernel software, between the IP protocol and the X.25 protocol. In order to maintain reasonable throughput, the IP to X.25 address mapping table must also be kept in the kernel where it is easily accessible to the interface. The drawback lies in the fact that the table must be kept to reasonable size to prevent it from using too much of non-pageable kernel memory. This effectively limits the number of hosts that can be reached through the interface to the number of address mappings the system can hold in the kernel address mapping table.

An alternative to the address mapping table approach is to use one of a number of possible two-way mathematical mappings between the IP and X.25 address spaces. Since the X.25 address space is several orders of magnitude larger than the IP address space, some sections of at least the X.25 address space will be invalidated by any such mathematical formula. However, this problem is not restricted to the formula approach since any translation method used has the same limitation in that some section of at least the X.25 address space will be left unused, even if only in an implied sense. The advantage of the formula approach is that IP to X.25 address translation can be performed dynamically in both directions with negligible space consumption. If the formula and the resulting algorithm are simple enough, the computational cost could also be made to approach that of the table lookup operation needed with the address mapping table approach, and the win would be substantial.

However, to be fair, the formula approach makes some strong assumptions regarding the administration of both address spaces which can make it completely unfeasible in certain situations. This point will become apparent when the DDN and PDN approaches to interfacing IP to X.25 are described in more detail.

3.2. X.25 Virtual Circuit Management

Another important issue to resolve when interfacing IP to X.25 is managing the available X.25 virtual circuits to effectively transmit and receive IP datagrams. Central to this issue is the fact that IP is a datagram protocol while X.25 is a virtual circuit protocol.

Following the nature of datagram protocols, IP will hand the interface a series of outgoing packets which have no explicit or implied locality of reference with respect to destination hosts. In practice, such locality of reference may very well take place and in fact be a common occurrence, but nevertheless the interface must be prepared to handle outgoing datagrams for any destination at any time. The interface cannot assume that any sequence of outgoing IP datagrams is destined for any one host or in fact for any particular subset of all the hosts in the network.

At the same time, X.25 provides only a limited number of virtual circuits over which to send such datagrams. The number of X.25 virtual circuits available to a host is further limited by the X.25 protocol implementations on both the host and network sides of the X.25 connection. This number of real simultaneous X.25 virtual circuits typically range from 32 to 128, certainly smaller than the number of hosts in a large internet and much smaller than the number of hosts addressable by IP and X.25.

Given the fact that X.25 virtual circuits must be explicitly established to a particular host before any data can be sent to that host, a non-trivial operation in time and in some cases money, X.25 virtual circuits must be considered a very scarce resource in the context of an IP to X.25 interface. It follows that effectively managing this resource plays a crucial role in the behavior of the interface.

The problem of managing a scarce resource to satisfy a much greater or theoretically infinite demand is common to many aspects of operating system design. It is then reasonable to attempt to draw from past experience in other areas to solve the specific case of X.25 virtual circuits and IP datagrams. In particular, this problem is in many ways similar to that of managing a limited amount of real memory to satisfy a much larger demand for virtual memory. It can be expected that some of the more successful schemes for virtual memory management would also work well here. Describing the many virtual memory schemes that have been developed is outside the scope of this paper, but suffice it to say that schemes such as preemptive Least Recently Used (LRU) algorithms and the working set concept apply very well to X.25 virtual circuit management in an IP to X.25 interface.

Aside from what algorithm to use to schedule the number of available X.25 virtual circuits among the number of destination hosts referenced by IP, a related decision is how many X.25 virtual circuits to open per destination host. Since an unavoidable delay is incurred between the time one particular X.25 virtual circuit accepts a packet for output and the time it can accept another one, it is logical to assume that opening more than one virtual circuit per destination host will help improve throughput. In particular, the diminutive window sizes used by X.25 networks to insure reliable transmission tend to prevent a single virtual circuit from approaching data rates anywhere close to the X.25 line speed [3].

3.3. Packet Fragmentation

As mentioned in the description of the IP protocol, part of the IP specification assigns IP the responsibility for insuring that the packets it hands lower level network interfaces do not exceed the maximum packet length allowed by that network. With X.25, there is a very clear notion of the maximum packet size being used over any particular X.25 link. The simplest way to insure that IP packets destined for an X.25 network are smaller than the maximum X.25 packet size for the link is to let IP use this maximum packet size as its maximum transmission unit for that particular IP to X.25 interface.

IP can this way handle the fragmentation and reassembly necessary for transmission over X.25 networks, but there are practical drawbacks to the approach of letting IP take care this task. For one, maximum packet sizes in X.25 network links run as low as 32 bytes, with many X.25 networks using 128 bytes as the default value. With packet sizes this small, the overhead of including a 20-byte minimum IP header with every X.25 packet becomes an important performance factor.

An alternative to IP fragmentation is to let the IP to X.25 interface perform its own fragmentation and reassembly. This alternative is available through the use of the X.25 Level 3 M bit, or "more data" bit, which can be used to implement packet fragmentation and reassembly with only a 3-byte X.25 header overhead per fragment. A tradeoff is involved since the number of transmitted bytes saved by not sending an IP header must be weighed against the extra processing and duplication of effort necessary in the IP to X.25 interface in order to process fragmented data.

3.4. Internetwork Routing

As explained when the IP protocol was described, one of IP's major contributions to the DARPA family of protocols is its ability to route packets through an internetwork of machines. In order to do this, IP relies on the partitioning of the IP address into a network part and a host part. For IP to route packets from one network to another, each network must be assigned its own network number.

Similar to the case of formula based address translation, however, the ability to assign different network numbers to different networks makes strong assumptions on the autonomy available in the administration of the IP address space. If certain constraints prevent two networks from being assigned two different network numbers, then IP will not be able to route packets between these two networks. This is because IP will not have the necessary information available to it in the source and destination address fields in IP packets. In this case, it is up to the IP to X.25 interface to perform some form of internetwork routing or IP packets routed through these interfaces will not be able to traverse more than one network.

Once again, as with the packet fragmentation issue, a tradeoff is involved in letting IP perform a job it is capable of doing versus duplicating the functionality in the IP to X.25 interface.

Together with some of the other general IP to X.25 interfacing issues discussed above, purely administrative factors will strongly affect what decisions are made to resolve these issues. Some of these factors will become apparent when the DDN and PDN approaches to IP to X.25 interfacing are described in the next section.

4. Two Approaches to IP to X.25 Interfacing: The DDN and PDN Standards

As outlined above, there exist several alternative solutions to two of the central issues to be resolved by IP to X.25 interfaces, IP to X.25 address translation and X.25 virtual circuit management. This is also true of a number of less general, sometimes network-specific choices that must be made when interfacing these two protocols. The freedom in many of these areas is such that different decisions taken by different implementations will very likely make them incompatible. Therefore, different IP to X.25 interfaces must agree exactly on many issues if they are to communicate.

For this reason, as is true with the protocols they serve to join, standards are necessary for different implementations of IP to X.25 interfaces to interact in a reasonable fashion. Two such standards have developed for two different types of X.25 networks, one for the X.25 portion of the Defense Data Network (DDN) and one for X.25 Public Data Networks (PDNs). They agree in many areas, but differ on their approach to some of the general design issues described in the previous section. They also differ on their support of several features specific to either the DDN or particular PDNs.

4.1. The DDN Approach

The Defense Communications Agency (DCA) has decreed that all new host connections to the DDN be made with X.25, and not with the more traditional 1822 ARPANET protocol. Indeed, X.25 is slated to completely substitute 1822 ARPANET in the DDN. Furthermore, if a host wants to interact with other DDN hosts using the full functionality available in the DDN, i.e., with the full range of DARPA-family protocols, then it must include an IP to X.25 interface.

To aid in this purpose, the DCA has published a definitive standards document, the DDN X.25 Host Interface Specification [2]. This specification dramatically narrows the number of implementation decisions faced by developers of IP to X.25 interfaces for the DDN.

In the case of address translation, as with other issues, the standard allows different implementations to interact successfully by exactly specifying the actions to be taken by all IP to X.25 interfaces in the DDN. For IP to X.25 address translation, hosts in the DDN must use two simple translation formulae which directly map certain fields in the IP address to other fields in the X.25 address, and vice versa.

On the issue of how many virtual circuits are opened between each host pair, only one X.25 virtual circuit is opened between any two hosts on the DDN. This certainly simplifies things, although it eliminates the possibility of added throughput between any pair of hosts through the use of multiple virtual circuits. However, as DDN X.25 links can operate near the upper limits of packet sizes, window sizes, and transmission speeds available in X.25 networks (1024 bytes per packet, window size of 7, and 56 Kilobits per second), the cost incurred is minimized.

Regarding internetwork routing, the DDN makes full use of the internetworking features of the IP protocol. This relieves the interface from having to resolve addresses for different physical networks in order to route packets among these networks.

For example, consider the case when data is destined for a host on a network to which the local host is not directly connected, but a host on the directly connected X.25 network exists which can act as a gateway to the destination network. On the DDN, IP will be configured with the necessary routing information to build routing tables that include the gateway address as the address to send all packets destined for the destination network. IP will build a packet with destination address as normal, but will hand the interface the gateway address as the immediate destination address for the packet instead of the eventual destination address. Following correct protocol layering procedures, the interface does not look inside the IP packet and is therefore unaware that the gateway is not the final destination. It proceeds to blindly send the packet to the gateway host.

The network interface on the gateway host will be just as ignorant of the discrepancy in destination addresses and will pass the incoming packet to IP. It is only in the IP layer of the gateway host that the IP packet is again looked at to determine that the destination host is not the gateway host. The packet will be once again routed to either the final destination or another gateway from information maintained exclusively by the IP layer. This constitutes correct usage of the internetwork addressing and routing capabilities designed into IP.

Aside from the generic design issues explained above, there are several network-specific features which an IP to X.25 interface to the DDN must include. One of the most visible examples is the specification of DDN standard service. X.25 hosts on the DDN can subscribe to either basic or standard service. DDN basic service can be thought of as raw X.25 service provided by the network for host-to-host communication on one physical X.25 network. Basic service does not imply the use of an IP to X.25 interface to access the network. More relevant to this discussion, DDN standard service is oriented towards DDN X.25 hosts using the standard TCP-IP higher level protocols. IP to X.25 interfaces on the DDN must clearly specify DDN standard service by means of a private or non-CCITT facility[3] included in all outgoing X.25 call request packets. Similarly, these interfaces should expect this facility on incoming X.25 call requests. Other network-specific features are similarly made available through the use of non-CCITT facilities, for example DDN call precedence.

4.2. The PDN Approach

In the case of Public Data Networks, a detailed standards document in the fashion of the DDN X.25 Host Interface Specification is not available. There does exist, however, a Request For Comments (RFC) from the DDN Network Information Center (NIC), RFC 877 titled "A Standard for the Transmission of IP Datagrams Over Public Data Networks" [5]. An excellent complement to this document is the source code release of Purdue University's X.25 Network Interface (XNI) software for the X25net portion of the Computer Science Research Network (CSNET), described in [1]. This software can be considered a de facto standard for interfacing IP to X.25 for operation over a PDN. It is available to CSNET members directly from the CSNET Coordination and Information Center (CIC).

The address translation formulae used in the DDN are convenient and efficient, but their use assumes that the network administration entity has complete control of both IP and X.25 address spaces. This is true in the DDN where a central organization, the NIC, is responsible for assigning both IP and X.25 addresses to all DDN X.25 sites. They can simply assign any available IP address and then use the IP to X.25 formula to assign a corresponding X.25 address, or vice versa.

Unfortunately, in the PDN case the network administration is not so centralized. Typically, the X.25 address space is controlled by the PDN administrating agency, but the IP address space is still controlled by the NIC. In the case of a logical network like CSNET, the CSNET CIC does not control either address space. They are forced to rely on the PDN carrier for assignment of X.25 addresses, and on the NIC for assignment of IP addresses. Because of these practical limitations, IP to X.25 interfaces for PDNs must use an alternative method to the formula approach.

As described in Section 3, using an address mapping table is a simple and effective way of performing IP to X.25 address translation. Such a table is used by the CSNET XNI software. This table contains one entry for each host that the interface needs to communicate with, including the local host. Each entry in the table contains a host name, its IP address, its X.25 address, and other PDN-specific information for the host. IP to X.25 address translation is then possible by performing a table lookup operation.

[3] X.25 facilities are special byte codes inserted in X.25 control packets in order to specify that certain non-default options are desired in the current virtual circuit session.

On the issue of how many virtual circuits are opened between each host pair, RFC 877 and the CSNET XNI software allow for several X.25 virtual circuits to exist between a pair of hosts. This make possible added throughput between any pair of hosts by demultiplexing data over multiple virtual circuits. This is a great advantage in the PDN context, where the packet sizes, window sizes, and transmission speeds used are typically small enough (128 bytes per packet, window size of 2, and 9.6 Kilobits per second) to cause significant delays in IP datagram transmission. How many virtual circuits to open to a remote host is specified in the address mapping table entry for that host.

Because of practical network administration limitations similar to those encountered with IP to X.25 address mapping, the CSNET XNI software is forced to take on many of the internetwork routing responsibilities usually associated with IP in order to operate successfully in an internetwork environment. The interface must resolve addresses for different physical networks in order to route packets among these networks, and maintain enough information within itself to make proper use of any available gateways. The network interface in the gateway hosts must also take on additional routing responsibilities since IP is not being used for this purpose.

For example, CSNET spans several physical networks and includes gateways to many more [6]. However, the CSNET administration is forced to assign its member hosts IP addresses with the same IP network number regardless of which physical network they reside in. An IP to X.25 network interface connected to CSNET over an X.25 PDN must perform its own internetwork routing if it is to transmit data to CSNET hosts in networks other than its own PDN. For this purpose, hosts not in the same PDN one are explicitly marked as such in the address mapping table entry for that host. Gateway hosts are similarly marked. With this information, IP to X.25 interfaces can route all packets to a remote PDN through an appropriately marked gateway, and the gateway hosts can subsequently continue the internetwork routing process. This is exactly equivalent to the routing performed by IP in the DDN case, but the information used to perform this function is contained solely within the IP to X.25 interface and not in IP.

Aside from the generic design issues explained above, there are several network-specific features which an IP to X.25 interface to PDNs must include. An important example is the specification of reverse charging, i.e., collect calls. Each X.25 call request can be made using either direct charging or reverse charging. If desired, IP to X.25 interfaces on PDNs can specify reverse charging by means of a standard CCITT facility included in outgoing X.25 call request packets. Similarly, these interfaces should be prepared to handle this facility on incoming X.25 call requests from other hosts. Other network-specific features are made available through the use of facilities, both standard and non-standard. Examples of standard facilities are packet size, window size, and throughput class negotiations. Examples of non-standard facilities are non-CCITT packet size and window size negotiations still used in some networks as historical remains of implementations that follow pre-1980 CCITT recommendations.

5. Merging the DDN and PDN Approaches

Designing and implementing an IP to X.25 interface involves solving many generic design problems, some of which were explained in Section 3. Many of these issues have already been resolved for the implementor by one or both of the DDN and PDN standards described above. However, creating an interface that conforms to both the DDN and the PDN standards introduces additional design and implementation issues. Pyramid Technology Corporation recently undertook this task, and merged the two approaches in a single IP to X.25 interface that is able to operate over both the DDN and a number of PDNs.

Which of the two modes the interface operates in is dynamically configurable by the system administrator through the notion of an "interface type." All IP to X.25 interfaces in a system, each corresponding to one physical X.25 network connection, can be individually configured as being of either DDN or PDN type from software available to the system administrator. Thus, one Pyramid system can operate several IP to X.25 network interfaces over separate X.25 network connections, each conforming to either the DDN or the PDN standard.

A principal design goal of the Pyramid effort was to keep any duplication of functionality within the interface to a minimum. This avoids creating what in effect would be two separate interfaces, each highly specialized to either the DDN or the PDN case. On the other hand, it is also important to clearly delineate what functions of the interface are generic, what portions are DDN-specific, and what portions are PDN-specific. This is so both DDN and PDN standards can be easily tracked as they evolve.

The solution arrived at was to write the main body of the interface as a set of generic routines which handle all the functionality common to both DDN and PDN modes of operation in one single piece of software. These routines run regardless of whether the interface is configured for DDN or PDN, with checks for interface type made where non-generic operations are needed. At these checkpoints, DDN-specific or PDN-specific routines are called to perform the necessary network-specific functions. Thus Pyramid's IP to X.25 interface is broken into three well-defined portions: a generic main module, a DDN-specific module, and a PDN-specific module. This software structure is apparent in Figure 2.

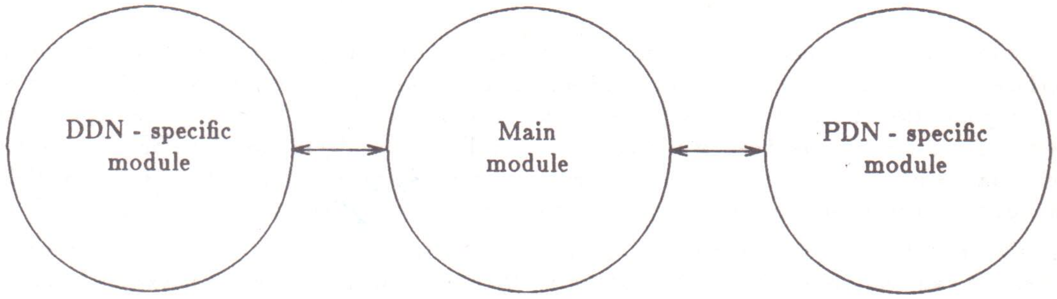

Figure 2 - IP to X.25 Interface Software Structure

As was hoped, the main module is able to handle a large portion of the necessary work, and the DDN-specific and PDN-specific modules are only a fraction of the size of the main module. The DDN module contains routines for IP to X.25 address translation according to the DDN formulae, as well as routines for allocating and using only one X.25 virtual circuit to any particular remote host. In contrast, the PDN module contains routines that create and maintain an IP to X.25 address mapping table. The information contained in this table is used by other PDN-specific routines to perform IP to X.25 address translation, and to allocate and use more than one X.25 virtual circuit to any particular remote host. There are also other network-specific functions handled among the three modules, for example the parsing and generation of DDN-specific and PDN-specific X.25 facilities.

6. Implementation Details

Aside from the design decisions faced by the Pyramid IP to X.25 interface effort, there are several interesting implementation issues that merit some discussion.

6.1. Asynchronous Finite State Machines

First is the question of how exactly to handle the task of sending and receiving individual IP datagrams to a specific remote host over X.25 virtual circuits, on which a certain sequence of events must take place before data can be transferred or received.

The Pyramid interface handles this problem through an IP to X.25 finite state machine, whose state transition diagram is pictured in Figure 3. Each available X.25 virtual circuit is considered to be in one of six states: FREE, CLEAR, CALLWAIT, CLEARWAIT, ESTABLISHED, and FREEING. FREE is the pristine state where a virtual circuit is unused and ready for allocation to a particular remote host. A virtual circuit in the CLEAR state is idle, either immediately after allocation to a remote host or immediately before it is freed.

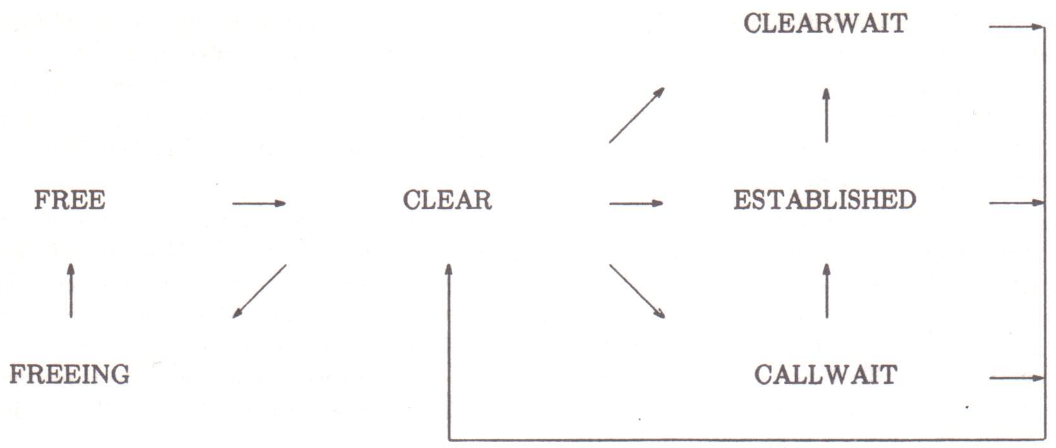

Figure 3 - IP to X.25 Finite State Machine

When the local host considers that a new virtual circuit is needed to a remote host, the interface finds a FREE virtual circuit, places it in a CLEAR state and writes a call request on it. At this point the virtual circuit is placed in the CALLWAIT state until a call accept packet is received from the remote host, when the circuit will be considered ESTABLISHED and higher level data transfer can take place. An alternative to receiving a call accept packet is receiving a clear request packet, upon which the virtual circuit goes directly back to the CLEAR state.

On receipt of call request packets from remote hosts, a FREE virtual circuit is allocated to answer the call and placed in the CLEAR state. If the incoming call request packet is legal, a call accept packet is written on the virtual circuit, which is placed directly in the ESTABLISHED state. On the other hand, if the incoming call is to be refused, a clear request packet is written on the virtual circuit, which is placed in the CLEARWAIT state until a clear confirmation packet is received. The circuit then goes back to the CLEAR state. If at any time during the CALLWAIT or ESTABLISHED states a clear request packet is received, the virtual circuit is placed directly in the CLEAR state.

With the exception of virtual circuits just allocated from the FREE state, virtual circuits placed in the CLEAR states are quickly placed in the FREEING state while the kernel and controller resources for that virtual circuit are released. When the resources have been freed, the virtual circuit is once again taken to the FREE state, ready for reallocation.

A lower level finite state machine is used to obtain and release virtual circuit resources on the underlying hardware controller board. Before any of the above higher level X.25 actions can take place, the interface must insure itself a set of controller resources corresponding to one X.25 virtual circuit. This is necessary for transmitting and receiving *any* data over an X.25 virtual circuit, including for example the initial call request packet. The steps necessary to reserve and release virtual circuit resources on the controller board are done in a separate state machine since, once this relatively simple and more synchronous sequence is finished, they become completely transparent to the subsequent higher level X.25 operations carried out by the IP to X.25 finite state machine.

6.2. Avoiding Sleeps in Interrupt Mode

A more UNIX-specific issue involves the fact that the Pyramid IP to X.25 interface is written as a BSD UNIX network interface, operating under the socket paradigm. The routines in these network interfaces are often called on hardware clock interrupts as higher level protocol timers go off. Together with the fact that network interfaces must also handle lower level hardware interrupts from the network controller, it follows that network interfaces very often operate in interrupt mode. Running in this mode introduces an important limitation: the kernel sleep and wakeup mechanisms cannot be used. These sleep and wakeup primitives are very convenient for synchronization within the kernel.

This limitation poses some interesting problems for the IP to X.25 interface, as schemes traditionally used in software of this type to wait and act on hardware controller interrupts are not usable. The most apparent change in software structure is the forced migration of functionality to the interrupt service routine, which further increases the amount of work done in interrupt mode. All in all, the interface must be very careful to insure that sleep is not used in any section of software that can run in interrupt mode. The same applies to using other routines that may themselves use the sleep mechanism, as is common with kernel resource allocation routines.

To eliminate the worry of possibly using sleep in an interrupt context, as well as for consistency, the Pyramid IP to X.25 interface completely does away with using sleep or any routine that may use sleep, even in those sections guaranteed to run only in a normal process context. Instead of synchronously waiting for expected interrupts in line with I/O requests, I/O requests are issued and control is always immediately returned to higher levels without waiting for the ensuing interrupt. Interrupts are then handled asynchrounously as they come in, with any additional processing remaining on the I/O performed under control of the two asynchronous finite state machines described earlier. In retrospect, this was a very advantageous early decision. Once the software was structured not to rely on the sleep and wakeup primitives, handling controller interrupts was reduced more to a problem of conforming to a particular coding style than to solving special cases of kernel-controller synchronization.

6.3. Interacting with Raw X.25

An important Pyramid-specific constraint on the implementation was the need for the IP to X.25 interface to closely cooperate with an existing X.25 implementation. This implementation consists of a UNIX character device interface to a General Purpose Synchronous Communications (GPSC) hardware controller, on which X.25 levels 1 through 3 run. A crucial goal for the Pyramid IP to X.25 interface was thus to successfully operate without disturbing the existing raw character device interface to the X.25 protocols on the hardware controller.

This goal was succesfully met with minimal changes to the raw X.25 interface by having the IP to X.25 interface interact with the controller in exactly the same way a user of the character device driver would. Thus, for example, the IP to X.25 interface requests and releases resources on the controller using the same sequence of commands a user of the existing character device interface would. A similar statement applies to performing the operations basic to any UNIX device: open, read, write, ioctl, and close.

Because of the no-sleep requirement described above, the existing device driver could not be used as is and a parallel device driver was developed which closely duplicates the behavior of the original. The major differences between the two are simply that this parallel driver does not use the sleep and wakeup synchronization mechanisms, and that it is designed to take data from and pass data to kernel network interfaces and not to user processes. Through the use of this no-sleep device driver, changes to the original device driver were reduced to the addition of a trivial check in the interrupt service routine in order to pass certain interrupts to the no-sleep device driver instead of processing them as is normal for the character device driver. The relationship between the network interface, the no-sleep device driver, and the original character device driver is shown on Figure 4.

6.4. Multiprocessor Support

A final Pyramid-specific implementation issue involves running the IP to X.25 interface in a multiprocessor system. Pyramid OSx is a symmetric multiprocessor operating system, and any piece of kernel software such as the IP to X.25 interface must be able to operate in the multiprocessor environment available in Pyramid hardware.

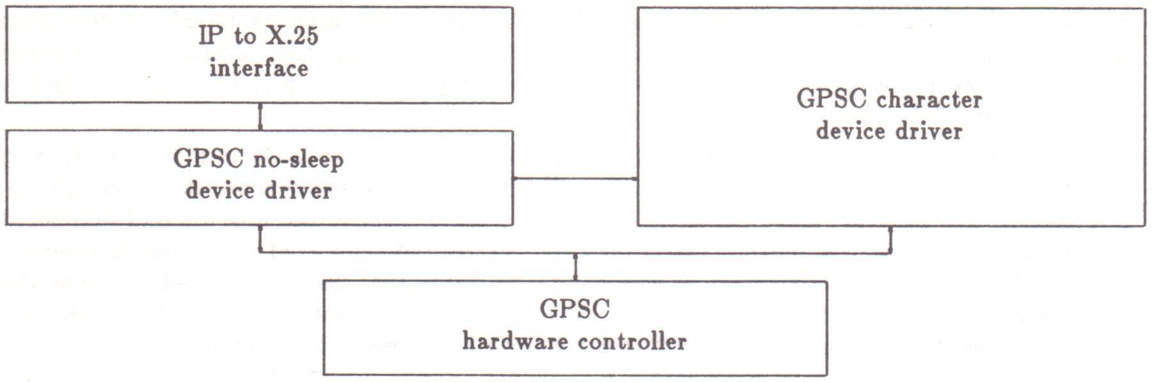

Figure 4 - Relationship Between Network Interface and Character Device Driver

More specifically, critical code sections and data structures must be protected from re-entry and modification by other processors since kernel software cannot be assumed to be running exclusively in any one processor for any length of time. In OSx, these critical sections are handled through explicit calls to multiprocessor semaphore locking and unlocking primitives. This is roughly analogous to the way critical sections and data structures in a sigle processor environment are protected against re-entry and modification by interrupt service routines through explicitly raising and lowering the interrupt priority level of the processor.

The IP to X.25 interface uses two levels of these multiprocessor semaphore locks. A per-interface lock is used when changing or testing per-interface global information, such as the address mapping table. Per-virtual circuit locks are also used when changing or testing per-virtual circuit information, for example the virtual circuit state. Other locking conventions established elsewhere in the kernel are used as needed when changing or testing non-interface-specific data structures.

7. Future Work

There are several interesting issues for further investigation in the area of IP to X.25 interfaces, and the Pyramid implementation in particular. Most of these have to do with improving the performance of the current implementation now that the feasibility of its operation has been proven.

On the issue of fragmenting higher level data inside the IP layer or inside the interface itself, it would be beneficial to study the effect on performance of varying the degree of fragmentation performed by IP and that performed by the interface. The results of these tests can be used to develop an effective policy for managing data fragmentation. Mainly, it would be very advantageous to find out under which X.25 packet sizes, window sizes, and transmission speeds it pays to avoid the IP header overhead and use X.25 M bit fragmentation, and under what conditions it is better to incur the IP header overhead but avoid the processing necessary for X.25 M bit fragmentation and reassembly.

On another topic, the address mapping table is presently used exclusively by the PDN-specific module. The information in this table is never used when operating over the DDN, when formula-driven address translation and simple linear search over the active virtual circuits is used for IP to X.25 address translation and virtual circuit allocation, respectively. Given that access to the address mapping table is highly optimized through the use of a hashing function for both IP to X.25 address translation and virtual circuit allocation, it may be advantageous to use the address mapping table on DDN-type interfaces as well. Further work on migrating the address mapping table handling to the main body of the interface, and then using it as an address mapping cache for DDN interfaces, could yield some important performance enhancements.

Further performance improvements, specially during high load conditions, can also be obtained by carefully tuning the virtual circuit timer policies used for LRU preemption of virtual circuits. Finally, at a lower implementation level, the IP to X.25 software could certainly benefit from detailed performance analysis though the use of kernel profiling and other methods.

Conclusion

This paper presents the Pyramid IP to X.25 interface, which connects the DARPA family of protocols to both the X.25 portion of the DDN and to a number of PDNs. Several aspects of this network interface are discussed, from overall design decisions generic to all such interfaces to some of the more interesting implementation issues faced during its development.

An IP to X.25 interface such as the one described here enables a host to access X.25 networks with the full functionality of the DARPA family of communication protocols, including virtual terminal sessions, bulk file transfers, and electronic mail delivery. This allows the host to use these facilities over a wide area network, an option that traditionally has not been available to large sections of the computer community. Together with the internetwork capabilities of IP and those of the network interface itself, outstanding wide area to local area network connectivity is also made possible. Add to this compatibility with both of the standards that have developed for IP to X.25 interfaces, DDN and PDN, and the software described in this paper constitutes a powerful networking tool. All in all, this network interface is an important addition to Pyramid's communications capabilities.

Acknowledgements

I would like to thank John Mahr for originally suggesting this paper, and Earl Stutes for testing the IP to X.25 interface over various network configurations. I would also like to extend special thanks to Rick Kamicar for his timely help on formatting this document. Finally, posthumous thanks must once again go to Bob Marley for his continued musical support.

References

[1] Comer, Douglas E., and Korb, John T., "CSNET Protocol Software: The IP-to-X.25 Interface," *Proceedings of the Symposium on Data Communications*, ACM SIGCOMM, March 1983, pp. 154-159.

[2] *Defense Data Network X.25 Host Interface Specification*, Defense Communications Agency, Washington D.C., December 1983.

[3] Denning, Peter J., Hearn, Anthony, and Kern, C. William, "History and Overview of CSNET," *Proceedings of the Symposium on Data Communications*, ACM SIGCOMM, March 1983, pp. 138-145.

[4] *Internet Protocol Transition Workbook,* Network Information Center, SRI International, Menlo Park, California, March 1982.

[5] Korb, John T., "A Standard for the Transmission of IP Datagrams Over Public Data Networks," RFC 877, *DDN Protocol Handbook*, Volume Three, DDN Network Information Center, SRI International, Menlo Park, California, December 1985.

[6] Quarterman, John S., and Hoskins, Josiah C., "Notable Computer Networks," *Communications of the ACM*, Volume 29, Number 10, October 1986, pp. 932-971.

[7] *The X.25 Protocol and Seven Other Key CCITT Recommendations*, Lifetime Learning Publications, Belmont, California, 1981.