On the Use of Directory Services to Support Multi-Protocol Inter-operability[*]

Russell J. Clark
Kenneth L. Calvert
Mostafa H. Ammar

GIT-CC-93/56

September 20, 1993

Abstract

Multi-protocol systems are a vital tool for achieving inter-operability in today's heterogeneous communication networks. An important aspect of these systems is the need to determine which of the multiple available protocols will be used to carry out a given communication task; an uninformed choice can result in failure to communicate when communication should be possible. In this paper we consider ways to make information about hosts' supported protocol configurations available through directory services. We discuss various representation approaches, and describe a working implementation of a multi-protocol application exemplifying our approach.

College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280
rjc@cc.gatech.edu

[*]This research is supported by a grant from the National Science Foundation (NCR-9305115) and the TRANSOPEN project of the Army Research Lab (formerly AIRMICS) under contract number DAKF11-91-D-0004.

1 Introduction

The past decade has seen the development and deployment of many different protocol architectures, including TCP/IP, SNA, DECNET, and IPX. Radical growth in the number of interconnected systems has accompanied this proliferation of protocol suites. As a result of these two facts, there is an ever-increasing need to interconnect systems that do not currently use the same communication protocols. In a previous paper [3] we discussed support for multiple protocol suites as a method of achieving inter-operability in current and future networks. Such an approach is based upon the observation that universal support for any single architecture or even protocol is unlikely. The basic idea is to embrace heterogeneity by developing frameworks for dealing with it. This paper presents one element of such a framework, namely a method enabling hosts to obtain information about the protocols supported by other hosts.

Multi-protocol networking has sparked research in several areas. Cypser [4] describes three main locations for protocol switching in systems supporting multiple protocols. Ogle et al. [11] are developing a TCP/IP and SNA system that performs protocol selection below the socket level interface. Janson et al. [5] consider options for inter-operability between OSI and SNA networks, and analyze the addressing issues arising when these protocols are combined in a single network. Instead of an architecture-specific solution, however, we seek an approach general enough for problems involving future architectures as well as today's. Among others considering the general inter-operability problem, Tschudin has described a "generic protocol" [18] allowing communicating systems to exchange descriptions of arbitrary protocols before using them to communicate. Similar in philosophy is the "meta-protocol" concept proposed by Meandzija [8]. In contrast, one of the explicit goals of our work is to minimize the number of things (including protocols) that must be universally agreed upon or supported.

This paper considers the use of *directory services* to provide information about the various protocols that hosts support. Such information is useful in a multi-protocol context because there may be more than one protocol configuration supporting a given communication service, and choosing the wrong configuration can lead to a failure to communicate. Using directory services to provide this information at runtime is a natural extension of the primary use of these services today, i.e. mapping names to address information. We describe a generic approach, along with a working implementation based on a widely-used directory service, the Internet Domain Name System [9, 10].

The paper proceeds as follows. In the next section, we describe the problem in more detail, along with the basic idea of the solution. In Section 3, we consider possible methods of encoding the required information in a directory database. General features of a directory service supporting this function are discussed in Section 4. In Section 5 we consider ways to implement these features using the existing Internet Domain Name System. A working multi-protocol application, incorporating an implementation of the proposed features, is described in Section 6. Finally, Section 7 contains some concluding remarks.

2 The Problem

We consider an environment consisting of hosts connected by communication networks. The communication subsystem in each host supports some set of network protocols; different services are implemented by different combinations of these protocols. Protocols are grouped in *suites* or *architectures*; those in the same suite were designed to work together to provide certain services, while those in different suites may or may not work together. Note that two hosts need not support the same set of protocols, even if they both support the same (one) protocol suite. For example, in the OSI suite, some hosts provide connection-oriented network service via X.25, while others support only the Connectionless Network Protocol (CLNP).

2.1 Access to Communication Services

We assume that the communication subsystem provides services to users via an abstraction that hides details of the protocols from the user and serves as an endpoint for one communication instance, such as a connection. For lack of a better name, we shall refer to this abstraction as a *session*. The session contains information that determines which protocols process outgoing messages sent by a user, and that enables incoming messages from the network to be routed to the user. In addition, it may contain (pointers to) local state information required by the protocols.

A *protocol entity* (PE) is an abstract representation of a part of the communication subsystem that implements a particular protocol. A convenient description of a protocol entity is: "a component that adds a header to outgoing messages, and thus affects inter-operability with other systems". Each session has associated with it a sequence of protocol entities, namely the protocol "stack" implementing the communication (Note that in practice, the session does not have associated with it complete, separate protocol implementations, but rather capabilities to access the implementations, which are shared across sessions. The distinction is immaterial here.) The sequence of PEs associated with a session is here called a *path*. While in most architectures every message associated with a session is handled by the same path, this need not always be the case. In the architecture of [12], for example, the PE structure associated with a user session is actually a graph, and different messages can traverse different paths through this subgraph. Use of the term "path" notwithstanding, our model does not exclude such architectures.

The general scenario by which a user obtains network services through a session object involves the following steps:

1. **Service Determination.** Determine the *type of service* needed by the user. Note that "service" here may include such aspects as the format of the address by which the destination host is identified.
2. **Path Determination.** Determine the combination of protocols required to provide the needed service.
3. **Path Configuration.** Create a session object with the required protocol path, and pass it to the user.
4. **Communication.** The user performs the appropriate operations (open connections, send, receive, etc.) via the session.

Note that service determination may be performed by the user, or by some combination of user and communication subsystem. For example, the user may know the name of a desired destination, but not its address (or even its address format). To obtain the required information, either the user or the communication subsystem may make use of a *directory service*, which maps names to network addresses. Such a directory service may be a simple file local to the host, or a separate remote service, accessed via the network itself.

The protocol determination step is typically performed by the communication subsystem; however, it could be handled by the user *if* the subsystem allows the user to request a session object with any arbitrary protocol path.

A well-known example of an environment following this model is the Berkeley UNIX *socket* interface. The socket abstraction provides users with uniform access to network protocol (and other IPC) services, while the socket data structure itself contains the information that determines the path used for communications via the socket. When requesting allocation of a socket, the user must specify both a *socket type* (e.g. stream, sequence of packets) and an *address family* (e.g. Internet, OSI, etc.). The former generally determines the transport-level protocol used (e.g. TCP for streams, ISO TP4 for sequence of packets); while the latter determines the network-level protocol used (e.g. IP).

The fact that different services require different combinations of protocols implies that the binding between (at least some) layers is delayed until the time of the user request. That is, some *switching* mechanism is required to enable a layer N protocol entity to be configured to use different layer $N-1$ protocols for different sessions. Within a single protocol suite, typically only one path corresponds to any given combination of service and network address; thus the need for switching is minimal, and the mapping from services to paths *may* be fixed in advance by the communication subsystem. This is not necessarily true in multi-protocol environments.

2.2 Multi-protocol Complications

In general it does not make sense to combine protocols in arbitrary ways. There are at least two possible reasons why a PE a, at layer N, would be precluded from making use of another PE b at layer $N - 1$.

One reason is architectural: if b does not provide the service required by a for correctness, then it simply makes no sense for a to be configured to run on top of b. For example, it is not sensible to run the OSI Class 0 Transport Protocol on top of CLNP, because the former requires a reliable, connection-oriented network service, while the latter provides a datagram service.

The other possible reason is that a particular *implementation* may not support the configuration of a atop b. In earlier versions of Berkeley UNIX, for example, the use of IP as a network protocol was hardwired into the TCP implementation; such implementations do not support the use of CLNP by TCP, even though conventions for such use have been defined [2]. In some cases, this constraint can be overcome by the addition of a switching capability to the implementation, in the form of a "pseudo PE" that binds references to the actual lower-level PEs, and hides minor differences between their interfaces.

For a given set of PEs, the possible configurations that might be supported in a host can be represented by a directed graph with PEs as nodes, in which the presence of an edge from a to b means that a can be configured to run on top of b. In this case we say a *uses* b.

> **Note.** We are assuming here that selection of a lower-level PE is constrained *only* by the next higher-level PE. For example, if the paths $a \rightarrow x \rightarrow c$ and $b \rightarrow x \rightarrow d$ are both possible, then the path $a \rightarrow x \rightarrow d$ is also possible. This is tantamount to assuming that the service provided by a PE is independent of *which* lower-level PEs it uses, so long as it uses one with adequate functionality. One consequence of this is that the service provided to the user by a protocol path is fully determined by the topmost PE of the path. This assumption appears to be reasonable for existing protocols.

The graph representing the maximal *uses* relation consistent with the architectural constraints for a given set of PEs is called the *architecture graph*. That is, in the architecture graph there is an edge from a to b if and only if it is "sensible" for a to be used at some layer N with b as the layer $N-1$ protocol. An example architecture graph is shown in Figure 1, which indicates that FTAM can use either TP0 or TP4.

For a given host that supports a particular set of PEs, the *installation graph* is the subgraph of the architecture graph representing the *uses* relations actually supported in the local implementation. That is, the installation graph says what protocol paths are available to users on a given host. Given that the service provided by a protocol path is a function of the stack's topmost PE, and given a mapping from services to top-level PEs, the installation graph indicates which paths can be used to provide a given service.

It was noted above that a single protocol suite generally has a unique path for a given service–address combination. This ensures that two hosts supporting the same suite can inter-operate and offer the service to their users, provided they both support the required protocols; if one host does not implement some required protocol, the service cannot be provided in any case. In an environment where protocol suites are mixed in some hosts, the situation is more complicated. The architecture and installation graphs may contain multiple paths for the same service–address combination, and thus the Path Determination step involves selection of one of these paths. Moreover, installation graphs may differ from host to host; selection of the *wrong* path can prevent inter-operability even in the case where the hosts support some other common path.

2.3 Path Determination Approaches

We have identified two possible approaches to the problem of Path Determination in a multi-protocol environment. The *directory-based* approach makes use of a database of information about the installation graph supported by hosts on the network. The *path discovery* approach tries to establish communication using all of the possible paths, and monitors the results. This paper focuses on the directory-based approach.

Network directory services such as the Internet Domain Name System (DNS) or the OSI directory service (X.500) provide a distributed database of information about hosts, their addresses, and the applications they support. In current architectures this database is typically consulted to map host and service names to their respective network and application addresses during path determination. Adding information about a host's protocol paths to this database is thus a rather natural way to support path determination in a multi-protocol environment.

An important feature of the directory-based approach is that it does not require all hosts to make use of the directory service. For example, a host supporting only a single protocol suite need not refer to the directory's protocol path information at all, because its path determination problem is simple. To aid other multi-protocol hosts in establishing communication with the single-protocol host, information describing the protocol(s) it supports should be stored in the directory service, but no modification of its communication subsystem—or the way it uses the directory service, if any—is required. Note also that universal agreement on a single directory service is not required: it is only necessary that each multi-protocol host have access to a distributed database that contains *some* encoding of the protocol path information for the hosts with which it communicates. Nor is it necessary for every directory service to use the same encoding; different encodings might be used in different parts of the network. The only requirement is that users of a particular directory service agree on a standard method of encoding protocol path information. In the next section we consider some possibilities for such an encoding.

3 Encoding Protocol Path Information

We would like to encode information about a host's supported protocol graph in a form that enables another host to determine — at a minimum — whether a common path exists. Information about a host's supported protocol graph can be stored in a directory service in any of several forms. Such an encoding scheme requires that globally-understood identifiers be assigned to some parts of graph structure. There are at least three levels at which such identifiers might be assigned: Protocol Entity, Protocol Path, Protocol Graph.

To compare these approaches, we consider a host with the installation graph of Figure 1. This host includes four different application PEs, three different transport PEs, two network PEs and one physical PE[1]. The Figure also depicts the addition of a protocol switching function in three places: the Application Switch provides switching between the two OSI transport protocols, the TCP Switch provides TCP with the capability to select between IP and CLNP as a network service, and the TP Switch provides a similar function for TP4. These "pseudo-PEs" do not themselves implement protocols and do not add or modify message headers; they are included in the figure to emphasize that the protocols involved are designed to use a particular lower-level protocol, and do not support the switching function.

This graph supports ten different protocol paths: two for each TCP application and three for each OSI application. Let us consider the three protocol representation options as they would represent this particular example.

- **Protocol Entity:** The first option involves assigning identifiers to the PEs themselves. The installation graph structure can then be represented in any of several ways, e.g. by giving for each PE, a list of the PEs to which it has edges (its uses-list). The graph of Figure 1 would thus have an entry for "TCP" with a uses-list containing "IP" and "CLNP". The global identifiers are "TCP", "IP", etc. This approach can easily handle novel paths and configuration changes, and allows for partial matches when a complete matching path

[1]A more precise depiction of this graph would include OSI Presentation and Session layers between the FTAM and VT applications and the Application Switch. For simplicity in this discussion, we omit these here.

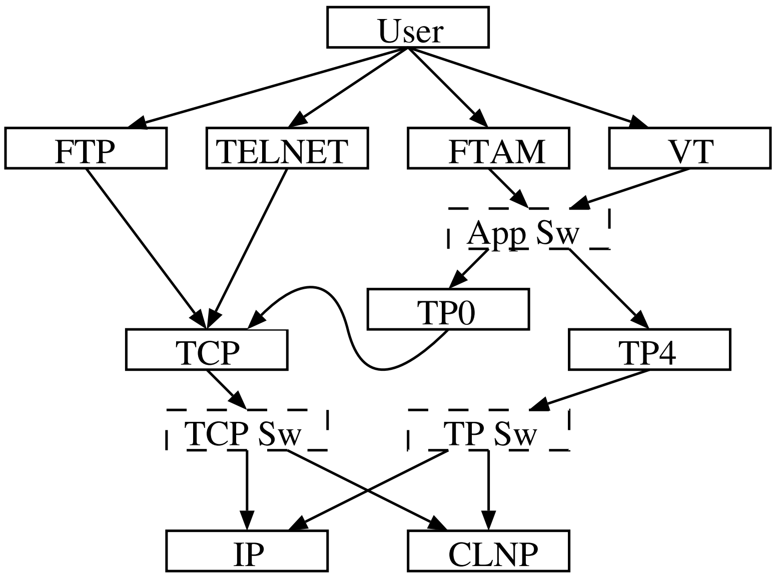

Figure 1: A multi-protocol graph.

cannot be found. Partial matches might be useful in establishing a minimal level of communication with an unknown system.

Another way to assign identifiers at the PE level is to encode the PE's uses-list (i.e. its outgoing edges) in the identifier. Thus an implementation of TCP that can use both IP and CLNP would have a different identifier than one that can only use IP. The graph could then be represented as a simple list of PEs. This would generally yield a smaller representation, at the expense of a larger number of agreed-upon identifiers.

- **Path:** The second representation option is to assign a standard identifier to every protocol path, and store a list of the paths supported, (rather than the installation graph itself). This approach simplifies path determination by providing a representation at the exact level where a match is sought. However, prior agreement on an identifier for every possible supported path is required; this may be a problem when new protocols and paths are introduced. This approach also provides a significant degree of redundant information because an application that uses multiple protocols will have a separate entry for each set of protocols it can use. For example, both the FTAM and VT applications in Figure 1 would be incorporated into three path entries.

  Note that if the set of paths cannot be characterized by the uses-list relations of the PEs involved, i.e. the assumption of Section 2.2 is not valid, then this option may be preferable.

- **Installation Graph:** In this option, the installation graph (or set of paths) itself is assigned a standard identifier. A significant problem with this representation approach is that it requires every existing protocol configuration to be known *a priori* so that a standard identifier can be assigned to it. This is infeasible given the diversity in the installed configurations of even single protocol suite systems.

From the above discussion, the protocol entity and path options appear to have the most potential. In order to more carefully compare the feasibility of the two approaches we examine the number of data items required to represent various protocol graphs. For the graph in Figure 1, assuming that a node's uses-list is encoded in its identifier, both options require the same number of stored items: ten PEs vs. ten protocol paths. When a new application is added using TCP, the PE approach requires the addition of a single PE entry. The path approach however, requires two additional path entries. If a new network layer PE were added, the PE approach would require the addition of one PE plus the addition or update of the transport PE that used this network PE. The path approach would require a new path entry for each application using this new protocol.

In general, the PE approach is more efficient whenever a host includes applications that can operate over multiple different protocols. Since multi-protocol systems are becoming more common, and indeed this is precisely the kind of system we are interested in supporting, the PE representation approach appears to be the most appropriate. As systems add more applications, the benefits of the PE approach are further realized.

4 An Ideal Directory Service

In this section we present the design requirements for a directory service that most effectively supports the path determination task. Our objective is to describe the necessary directory service features in a context which is free from the constraints of any current directory service products. Later we discuss how most of these features can be provided in a production directory service.

A problem in current directory service usage is the assumption that the availability of a particular network address for a host implies that the host supports a network protocol which utilizes that address. This assumption causes problems, for example, when translating gateways are used to provide transparent communication between two distinct protocols. In this scenario, the originating host must obtain an address for the destination that is compatible with the originating host's network protocol. For example, a host X which only supports IP cannot use a NSAP address to refer to another host Y even if X can communicate with Y through an IP/CLNP gateway. Host X will need an IP address to identify Y.

To address some of these issues [no pun], our multi-protocol directory service maintains network address information independent of protocol graph information. While it is true that before using a given network layer protocol it is necessary to obtain a network address for that protocol, the existence of a certain type of address for a system does not necessarily imply that the system directly supports any protocols which use that address.

Graph information is represented in the directory service as a collection of PEs and their uses-lists. The name of the PE is stored as a single string entry. The uses-list is stored as a string describing the set of PEs this PE can use. Conjunction and disjunction are indicated by the characters "&" and "|" respectively. Conjunction in a uses-list indicates that a PE requires the services of *both* underlying PEs to operate; e.g. OSI Presentation may require several Session Functional Units. Disjunction indicates that a PE can operate on any of the underlying PEs; e.g. the Transport Switch can select either IP or CLNP.

Table 1 presents the information desired in a directory service entry for the protocol graph shown in Figure 1. PEs with an empty uses-list are known as *base* PEs. These indicate that no lower layer matching information is available for paths that include this PE. In general, the network layer protocols will serve as base PEs.

PE Name	Uses-list
FTAM	TP0 | TP4
FTP	TCP
IP	
CLNP	
TP0	TCP
TP4	IP | CLNP
TCP	IP | CLNP
TELNET	TCP
VT	TP0 | TP4

Table 1: Protocol Graph Entry for a Multi-Protocol Host.

PE Name	Uses-list
FTP	TCP
IP	
TCP	IP
TELNET	TCP

Table 2: Protocol Graph Entry for a Single-Protocol Host.

The two main functions of a directory service for multi-protocol systems are:

- *LookupHost(input: Hostname, output: AddressInfo, GraphInfo)* This function retrieves the addressing and protocol graph information for the specified host from the directory service. The addressing information is returned as a collection of network addresses of various types. The graph information is returned as a collection of PEs with their uses-lists. The initiating host will invoke this routine once for the remote host and again to obtain its own local graph information.

- *MatchPath(input: GraphInfo, LocalGraphInfo, output: Path)* This routine compares the two graphs and returns one or more common paths. The overall goal is to find a protocol path that is common to both graphs and will provide communication between the user application and a base PE. The exact return value and algorithm used is dependent upon the ultimate goal of the multi-protocol system. Achieving each of the goals we consider is equivalent in complexity to solving the subgraph-isomorphism problem. While the only known solutions to this problem are intractable for large graphs, the limited size of the protocol graphs coupled with the focused goals outlined below makes it feasible to solve this problem as part of communication establishment. The three possible path matching goals are:

  - *Succeed or Fail:*
    If the user is only interested in obtaining communication or finding out if communication is possible then a function that simply finds and returns the first successful match would suffice. This algorithm should start by matching a single PE and then try to build a single matching path.

  - *All Matches:*
    If a user wishes to be able to choose from multiple possible paths then it is necessary for the function to find all matches between the two graphs and return them. This function would be useful when there are several protocols supported by both hosts but one may be more appropriate for the given application. It is also possible that one or more of the valid paths may be temporarily unavailable due to a network failure. In this case the multiple paths would allow the user (or application) to try several different paths until one succeeds.

  - *Partial Matches:*
    In some cases there may not be a complete match found from the application all the way down to the base PE. In this case it may be appropriate to return partial match information about the PEs that did match. This would allow the system either to obtain a degraded level of communication or provide meaningful diagnostics to indicate exactly which components of the protocol architecture are missing. Partial matches might also be used as an aid in determining which gateway or translating bridge services might be useful in obtaining the desired communication. The algorithm for finding partial matches should be able to start anywhere in the protocol graph and find all PEs that match between the two graphs. We will be exploring the use of partial matches for discovering gateway services as part of our future research.

Each of these preceding path matching goals focus on finding paths that allow hosts to communicate. These goals could be further qualified to find paths that provide a particular service. This limits the matching algorithm to a specific PE or set of PEs with which to start the search and for which a path is considered valid.

5 A DNS-Compatible Implementation

The Internet Domain Name Service (DNS) is a popular example of the type of directory service that could provide protocol graph information. In this section we present an approach to using DNS to provide this extended service. In Section 6 we describe our implementation of this approach.

Our primary objective in this design is to develop a mechanism for delivering multi-protocol information that provides as many of the features identified in Section 4 as possible while minimizing the impact on current directory service implementations. Our approach requires that additional DNS support be provided only in multi-protocol systems that will take advantage of the new DNS features. The changes we propose have no impact on systems that currently use the DNS directory services. An alternative approach to using DNS would be to extend an X.500 implementation such as QUIPU, which is available with the ISO Development Environment (ISODE) [14]. While this approach would give us more flexibility to define new host information records, the ubiquity of DNS in the current Internet makes it more suitable for providing a system which could be deployed today.

5.1 An Overview of DNS

The DNS, described in [9] and [10], provides a hierarchically distributed database of network host information. It is used primarily to provide hostname to network address resolution. The two main components of the DNS are the *domain server* and the *resolver*. The domain server provides name service within a DNS domain. A domain corresponds to an administrative group such as a company or university. The resolver generally runs on the client host and provides the lookup service by successively querying domain servers. The actual data is stored on the server hosts in text files known as *master files*. The basic unit of information stored in the DNS is a *resource record* (RR). Each RR includes, among other things, a NAME field representing the node to which this entry pertains, a TYPE field representing the type of information stored, and an RDATA field representing the actual data for this entry.

Some important types of RRs are: `A` — the host address, `MX` — mail exchange information, `WKS` — the supported well known services, and `TXT` — a free-format

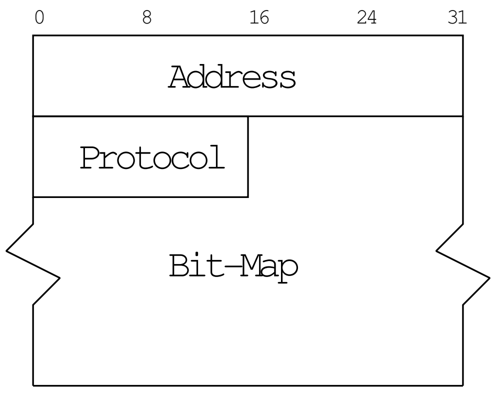

Figure 2: The WKS data format description.

text field. The WKS record format is described in Figure 2. This record has a 32-bit address entry indicating the IP address, an 8-bit entry indicating a protocol, and a variable length bitmap indicating which services use that protocol. The protocol field contains the identifier of a protocol that uses IP such as TCP or UDP. The bitmap indicates which of the well known services are supported on the host: if a service is supported then the appropriate bit is set. These well known service numbers are used as port identifiers in the TCP and UDP protocols. For example, if FTP is supported then bit 21 is set since FTP uses port 21 of TCP. The protocol and well known service numbers are defined in the Internet Assigned Numbers document [13].

The standard interpretation of the protocol field in the WKS record is that it represents a transport layer protocol such as TCP. The service bitmap represents direct users of the transport layer. This interpretation is consistent with the TCP/IP network model where application services sit directly on top of the transport protocol and the well known service number is used as the port identifier in the transport protocol. The organization of this record implies that any service listed will use the transport protocol listed for this record; an application using TCP should be listed in a separate resource record from an application using UDP.

5.2 A Multi-protocol Usage of DNS

While the DNS was developed primarily for the TCP/IP environment, it has evolved to accommodate heterogeneous networks. For diversity at the network layer, a number of address formats have been defined. These address formats include an X.25 format, ISDN format, and an OSI style NSAP format. These are stored as RRs of TYPE X25 , ISDN , and NSAP respectively. The RR type A is used for 32-bit IP addresses only.

An interesting aspect of the original design of DNS is the inclusion of a CLASS field in each resource record. This attribute is reserved for specifying information about the "supported protocol family" of a host [9]. The most natural extension of DNS to the multi-protocol environment is to use the CLASS field to designate which protocol architectures are supported. For instance, a class could be defined to indicate use of the OSI protocols. Unfortunately, this field has become largely meaningless in the current usage as only one value, "CLASS=IN" for Internet, has been widely used. Instead of designating different classes, each of the previously mentioned address type records has been created within the Internet class.

As we mentioned earlier, we are interested in developing a multi-protocol DNS that is compatible with most current DNS implementations. Our experience with current name server implementations, such as the BSD *named* program, is that they are largely hard coded for use with RR entries of class Internet. This means that the addition of a new CLASS value would require that current servers be modified to support the new classes and their associated types. We have decided not to pursue this approach since this change would conflict with our goal of not requiring the replacement of current systems.

We have identified three possible approaches to using the current DNS architecture for distributing multi-protocol host information. All of these approaches use the alternative described in Section 4 where identifiers are assigned to PEs. They all use the currently defined IN class resource records. The first two are based on the `WKS` RR and the third is based on the `TXT` RR. In the remainder of this section we compare these options and present our proposed solution. Each of the sample DNS entries presented in this section depict the master file format used by the DNS server to store the domain information.

5.2.1 A WKS Resource Record Approach

The first proposal involves extending the semantics of the `WKS` entries to allow any PE in a protocol graph to be represented in the protocol or service fields. Figure 3 shows a DNS master file entry for the host shown in Figure 1. Each line in this example corresponds to a separate RR. All of these RRs are associated with the host "mphost". The second field in each record indicates that the entry is of class IN for Internet. The third field indicates the type of RR data stored in this entry. The remaining fields contain the actual RR data. The first line in this example is the standard Internet address entry containing the IP address. The second line contains a host information entry describing the type of system associated with this host. In this case it is a Sun Microsystems Sparc 1 workstation running the SunOS operating system version 4.1. The third line is an NSAP type record providing an OSI-format address. This format is described in [6] and is currently being updated [7].

The remaining five lines of the sample DNS data are the `WKS` entries used to describe this host's protocol graph. Each entry specifies a PE and the set of PEs that use it.

```
; Name   Class   RR-Type   RR-Data
;-----------------------------          --------   --------   ----
mphost    IN     A       127.1.1.1
mphost    IN     HINFO    "Sparc  1" "SunOS  4.1"
mphost    IN     NSAP     49.5100bd5a00
mphost    IN     WKS      127.1.1.1   ip    tcp  iso-tp4
mphost    IN     WKS      127.1.1.1   clnp  tcp  iso-tp4
mphost    IN     WKS      127.1.1.1   tcp   ftp  telnet  iso-tp0
mphost    IN     WKS      127.1.1.1   iso-tp4    ftam  vt
mphost    IN     WKS      127.1.1.1   iso-tp0    ftam  vt
```

Figure 3: A multi-protocol DNS entry using `WKS` .

The fourth field indicates the IP address for this `WKS` entry. The fifth field specifies a PE and the remaining fields indicate the users of that PE. If a PE A uses a PE B then A will appear in an entry where B is in the fifth field. The only PEs that must appear in field five of a `WKS` entry are those that themselves are used by other PEs in the protocol graph. While this arrangement is natural for the traditional case where hosts have one or two transport protocols and multiple applications, it is exactly the reverse from what is needed in multi-protocol hosts where a single application may use multiple protocols.

The presence of the address field in the `WKS` entries causes some problems when using them with multiple protocols. One such problem can be seen in the second `WKS` entry where the IP address is given but a CLNP network protocol is listed as the PE. Because the address field has a fixed length of 32-bits, it is not possible to vary the address given to agree with the protocol specified. In general, our use of the DNS in this approach ignores the address in the `WKS` entries. Note also that we have provided explicit entries indicating the presence of the network layer protocols IP and CLNP. This is consistent with our goal outlined in Section 4 of keeping the protocol graph information separate from the addressing information. We do not assume that CLNP is available based on the presence of the NSAP address.

The overloading of the `WKS` semantics in this approach could be a problem for any current users of the `WKS` field and current server implementations. One server implementation, the BSD *named* program, makes explicit use of the *getprotobyname()* and *getservbyname()* routines to resolve the protocol and service fields as it reads the `WKS` entries from a master file. In UNIX systems these routines traditionally use the `/etc/protocols` and `/etc/services` data files to obtain assigned numbers for protocol and service names. Using additional PEs like CLNP in the protocols field (field five) will either cause problems in maintaining these data files or require that the name server be re-written to use an alternate data source.

5.2.2 A WKS Resource Record Approach With External Data

In order to avoid this overload of the protocol field in WKS entries we consider an alternate proposal using WKS RRs. In this approach, only PEs above the transport layer will appear in the bitmap portion (fields six and above) of the WKS entry. This alternative is characterized by the use of an external data source that provides additional meaning to the DNS entries. This data source could be maintained locally on the host or distributed by some other directory mechanism. We use this external information to map the PE entries in the fifth field to their uses-list. This maintains more consistency with the standard WKS usage by alleviating the need to store network layer PEs in the fifth field. A sample entry of this type is found in Figure 4. As in the previous approach, only those PEs that are used by another PE will appear in field five of a WKS entry.

```
; Name     Class    RR-Type    RR-Data
;------------------------------          ---------   ---------   -
mphost      IN      A        127.1.1.1
mphost      IN      HINFO     "Sparc  1" "SunOS  4.1"
mphost      IN      NSAP     49.5100bd5a00
mphost      IN      WKS       127.1.1.1   tcp   ftp  telnet
mphost      IN      WKS       127.1.1.1   iso-tp4    ftam  vt
mphost      IN      WKS       127.1.1.1   iso-tp0    ftam  vt
```

Figure 4: A multi-protocol DNS entry using WKS and external data.

The external definitions of the protocol and service identifiers from the WKS entry are given in Table 3. This is the additional data that must be provided to indicate the uses-lists of the WKS PEs. We assume that the "tcp" implementation supports either "ip" or "clnp" [2]. We also assume that the TP4 protocol "iso-tp4" uses either CLNP or IP. If this is not the case, another entry could be added with an assigned name such as "in-tp4" to identify a TP4 implementation that uses only IP. The entry "iso-tp0" represents the RFC-1006 mechanism for implementing TP0 over TCP [15]. Since TP0 and TP4 provide a similar interface, we assume that FTAM and VT can use either one. The identifiers "ftam" and "vt" are the only assigned names in this example not currently defined in [13].

Using both the WKS entry and PE definitions, a communication subsystem can build a protocol graph describing the available paths on the host of interest. The capabilities of this approach are the same as for the previous approach. The principle advantage of this approach is that it maintains greater compatibility with current WKS semantics for the protocol specifiers. This compatibility comes at the expense of requiring an additional data source and additional agreement on the meanings of the assigned names provided by that source.

Assigned Name	PE Name	Uses-list
ftam	FTAM	TP0, TP4
ftp	FTP	TCP
iso-tp0	TP0 w/RFC-1006	TCP
iso-tp4	TP4	IP, CLNP
tcp	TCP	IP, CLNP
telnet	TELNET	TCP
vt	VT	TP0, TP4

Table 3: Definitions of protocol and service identifiers.

5.2.3 A TXT Resource Record Approach

The final alternative we present uses the DNS `TXT` entries to encode protocol graph information. Currently, there is no standard describing the format of a `TXT` entry[2]. We propose that the `TXT` field be used to store a description of the PEs available on a host as well as the uses-lists of those PEs. The general format of the PE entries is "<PE>/<uses-list>". Figure 5 shows a grammar for parsing these `TXT` entries. We use the leading string "PEInfo" to distinguish these protocol descriptions from other `TXT` fields in use.

```
<TXT Entry>      ::=  "PEInfo:"<protolist>
<protolist>      ::=  <proto>";"<protolist>
<protolist>      ::=  <proto>
<proto>          ::=  <type>"/"<useslist>
<proto>          ::=  <type>
<useslist>       ::=  <uses>"|"<useslist>
<useslist>       ::=  <uses>
<uses>           ::=  <type>"&"<uses>
<uses>           ::=  <type>
<type>           ::=  [a-zA-Z0-9-]<type>
<type>           ::=
```

Figure 5: A sample grammar for parsing PEInfo `TXT` entries.

Figure 6 shows a possible DNS entry for a host with the installation graph given in Figure 1. Multiple PE entries in one `TXT` record are separated by a ";". If no uses-list is present, the entry is assumed to be a base PE. The entries are grouped by the protocol layer described. This grouping has no semantic meaning for the system and is strictly for the convenience of the person reading the file.

[2]Rosenbaum has proposed a new mechanism for using `TXT` fields for arbitrary string attributes [16]. At the time of this writing, this proposal had not advanced to the status of an Internet standard.

```
; Name   Class   RR-Type   RR-Data
;------------------------------            ---------    ---------    ---
mphost    IN     A       127.1.1.1
mphost    IN     HINFO    "Sparc  1" "SunOS  4.1"
mphost    IN     NSAP    49.5100bd5a00
mphost    IN     TXT     "PEInfo:IP;CLNP"
mphost    IN     TXT     "PEInfo:TCP/IP|CLNP"
mphost    IN     TXT     "PEInfo:TP4/IP|CLNP;TP0/TCP"
mphost    IN     TXT     "PEInfo:FTP/TCP;TELNET/TCP"
mphost    IN     TXT     "PEInfo:FTAM/TP0|TP4;VT/TP0|T       P4"
```

Figure 6: A multi-protocol DNS entry.

For our current implementation we have elected to pursue the use of the TXT fields to describe protocol graph information. This approach provides the most flexibility in encoding and presents the least danger of conflicting with current implementation and usage. One issue to note is that while the TXT RRs are defined in the specification [10] from 1987, they are not supported by all DNS implementations. The version of *named* found in the 4.3 Reno release of BSD UNIX includes support for TXT but the older BSD version that provides the basis for Sun OS 4.1.3 does not. This means that the WKS -based alternatives described above may indeed be more compatible with existing servers. However, since the use of newer address types like NSAP and X25 will require updating of name servers anyway, we expect that most Internet name servers will include support for TXT fields.

6 A Sample Implementation

As a proof of concept, we will now describe a working multi-protocol FTAM implementation capable of carrying out a file transfer with hosts supporting one of several different protocol architectures. The three protocol paths supported by this implementation are shown in Figure 7. The three paths are FTAM using: TP4 over CLNP, TP4 over IP, and TP0 using RFC-1006 over TCP/IP. These paths are also present in the host described in the DNS examples presented in the previous section. The basic steps carried out by the multi-protocol communication subsystem when attempting to provide the FTAM service are:

1. The user initiates a file transfer with the remote host by invoking FTAM with the remote host's name.

2. The communication subsystem obtains address and graph information by calling the DNS using the *LookupHost()* routine for both the remote and local

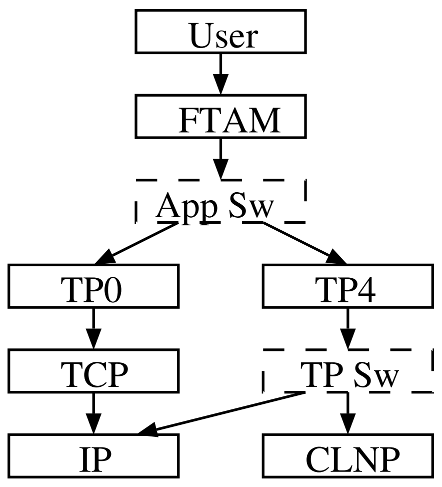

Figure 7: Multi-Protocol FTAM Architecture.

hosts.

3. The communication subsystem calls the *MatchPath()* routine to obtain a subset of the two installation graphs that includes the FTAM service.

4. The communication subsystem configures the local host's installation graph to use the selected path for this session[3].

5. Control is passed back to the user application (FTAM) which continues normally by establishing a connection using the routines configured by the communication subsystem.

Suppose that a user of this multi-protocol host wishes to perform a file transfer with the system "tcphost" described in Figure 8. This system only supports the single protocol path FTAM over TCP/IP using the RFC-1006 implementation of TP0. In this case, the only match found for performing file transfer is the "FTAM" entry. After matching the application, the next PE to match is "TP0", the only member of the "FTAM" uses-list for the host "tcphost". Next, "TCP" is matched and then "IP". Finally, once IP is selected, the A address entry is included for carrying out the communication.

To implement this system we developed a set of extensions to the BSD DNS resolver library and the ISODE 8.0 FTAM implementation. To the resolver we added the *LookupHost()* and *MatchPath()* functions discussed in Section 4. These extensions implement the path selection portion of the communication subsystem. The *LookupHost()* function retrieves the various address RRs and retrieves and parses the TXT fields to build the protocol graph information. *MatchPath()* compares the two protocol graphs and returns the first path found that supports the required service, which is FTAM in this case.

[3]Here we use our definition of session from Section 2 rather than the OSI Session.

```
; Name    Class   RR-Type   RR-Data
;------------------------------        ---------    -------
tcphost    IN     A       127.1.1.2
tcphost    IN     HINFO    "Sparc  1" "SunOS  4.1"
tcphost    IN     TXT      "PEInfo:IP;TCP/IP;TP0/TCP"
tcphost    IN     TXT      "PEInfo:FTAM/TP0"
```

Figure 8: A single protocol DNS entry.

Once the path is selected, the system performs path configuration by creating a session for the user which instantiates this path. In our implementation, this task is performed by the Transport Switch mechanism provided by the ISODE [14]. The Transport Switch provides a mechanism to select among protocol stacks based on address information. For each transport connection, it associates a separate structure of function pointers to transport layer functions. In turn, these particular transport functions invoke specific network layer functions. The ISODE implementation of the Transport Switch performs protocol selection based on the network address format of the destination host. If a CLNP style NSAP is found then the functions for implementing TP4 over CLNP are selected. If an IP style address is found then the functions for implementing the RFC-1006 TP0 over TCP are chosen.

To the ISODE switching architecture we added the ability to use the DNS routines to select which protocols should be used *without reliance on the address type*. In addition, we have added the ability to select TP4 over IP as one of the available paths. The use of this path is made possible by utilizing the protocol graph information from the directory service to select between two different paths, both of which use the IP network layer.

The ISODE includes an implementation of the Association Control Service in the form of the *acsap* library. This module provides the support for establishing a connection between the various Association Control Service Elements used by an application. It is within this library, specifically in the implementation of the routine *str2aeinfo()*, that we have added the multi-protocol DNS support. This routine takes a set of string descriptors, including a host name and service qualifier, and returns an application-entity information structure for this communication task. Included in this information is the address and protocol information utilized by the Transport Switch. An important result of this implementation approach is that it provides multi-protocol system support, not just to the FTAM application, but to all applications which utilize the *acsap* routines. In fact, there was no need to modify any of the actual FTAM application code itself. Furthermore, the current code architecture remains intact so that the regular ISODE databases can be used to resolve these entities if the information is not available from the DNS or the local administrator wishes to override the DNS information.

7 Concluding Remarks

In order to effectively use multi-protocol systems for communication in heterogeneous networks we must develop mechanisms for efficiently combining the protocol architectures and managing their use. Managing the use of these systems involves both determining which protocols should be used and then specifying that usage to the systems. This paper describes our work in using a directory service to aid in the determination task. We discussed techniques for extending current single protocol architectures to operate in a multi-protocol environment. We described a practical extension of the current DNS for multi-protocol systems that involves no modification to the currently deployed DNS server software. We also presented a successful implementation of a multi-protocol application capable of using these extensions.

It should be noted that the examples used in this paper deal with protocols and implementations that exist today. However, we expect that new architectures will be developed, and that the problems considered here will become even more important as the next generation of architectures are deployed. Current proposals for the next generation Internet protocols all involve some sort of transition strategy where both current and future protocols will need to co-exist [17]. Our proposed architecture is an appropriate solution to this multi-protocol co-existence problem.

The use of a directory service to solve this problem has some potential drawbacks that should be addressed. The usefulness of the directory service is limited by the accuracy of the directory information. If a host changes protocol graphs, by updating or replacing software, and the directory entries for that host are not correctly updated, other hosts will be unable to establish communication with that host. While current network hosts change protocol architectures no more frequently than they change addresses, recent research suggests that this may not always be the case in the future [12, 18].

The question of accuracy of the DNS entries has been raised in the Internet community. Indeed the Internet host requirements document [1] specifically warns that a host should not rely on the `WKS` entries to provide accurate information regarding the services available from another host. These concerns over accuracy lead to the likelihood that the directory service itself may not be enough to provide up-to-date information about a network host. Our future work will look at ways to combine the directory service with a more dynamic discovery system that determines the supported protocols from the host itself when the directory service information is incomplete.

References

[1] R. Braden. Requirements for internet hosts - application and support. *RFC 1123*, October 1989.

[2] R. W. Callon. TCP and UDP with bigger addresses (TUBA), a simple proposal for internet addressing and routing. *RFC 1347*, June 1992.

[3] R. J. Clark, M. H. Ammar, and K. L. Calvert. Multi-protocol architectures as a paradigm for achieving inter-operability. In *Proceedings of IEEE INFOCOM*, April 1993.

[4] R. J. Cypser. Evolution of an open communications architecture. *IBM Systems Journal*, 31(2):161–188, 1992.

[5] P. Janson, R. Molva, and S. Zatti. Architectural directions for opening IBM networks: The case of OSI. *IBM Systems Journal*, 31(2):313–335, 1992.

[6] B. Manning. DNS NSAP RRs. *RFC 1348*, July 1992.

[7] B. Manning and R. Colella. DNS NSAP Resource Records. *Internet Draft*, May 1993.

[8] B. Meandzija. Integration through meta-communication. In *Proceedings of IEEE INFOCOM*, pages 702–709, June 1990.

[9] P. Mockapetris. Domain names - concepts and facilities. *RFC 1034*, November 1987.

[10] P. Mockapetris. Domain names - implementation and specification. *RFC 1035*, November 1987.

[11] D. M. Ogle, K. M. Tracey, R. A. Floyd, and G. Bollella. Dynamically selecting protocols for socket applications. *IEEE Network*, 7(3):48–57, May 1993.

[12] S. W. O'Malley and L. L. Peterson. A dynamic network architecture. *ACM Transactions on Computer Systems*, 10(2):110–143, May 1992.

[13] J. Reynolds and J. Postel. Assigned numbers. *RFC 1340*, July 1992.

[14] M. T. Rose. *The ISO Development Environment User's Manual - Version 7.0*. Performance Systems International, July 1991.

[15] M. T. Rose and D. E. Cass. ISO Transport Services on top of the TCP. *RFC 1006*, May 1987.

[16] R. Rosenbaum. Using the domain name system to store arbitrary string attributes. *Internet Draft*, April 1993.

[17] IEEE Communications Society. Special issue: The future of the Internet Protocol. *IEEE Network*, 7(3), May 1993.

[18] C. Tschudin. Flexible protocol stacks. In *Computer Communication Review*, pages 197–205. ACM Press, September 1991.