Protocol discovery in multiprotocol networks *

Russell J. Clark [a], Mostafa H. Ammar [b] and Kenneth L. Calvert [b]

[a] *Empire Technologies, Inc., 541 Tenth St NW, Suite 169, Atlanta, GA 30318-5713, USA*
[b] *College of Computing, Georgia Institute of Technology, Atlanta, GA 30332-0280, USA*

Interoperability requires that communicating systems support compatible protocols. Maintaining compatible protocols is problematic in heterogeneous networks, especially in a wireless infrastructure where hosts can move from one protocol environment to another. It is possible to improve the flexibility of a communication network's operation by deploying systems that support multiple protocols. These multiprotocol systems require support mechanisms that enable users to effectively access the different protocols. Of particular importance is the need to determine which of several protocols to use for a given communication task. In this work, we propose architectures for a protocol discovery system that uses directory services and protocol feedback mechanisms to determine which protocols are supported. We describe the issues related to protocol discovery and present protocol features necessary to support multiprotocol systems.

1. Introduction

The communication network is quickly becoming a critical component of computer systems in both educational and commercial environments. The abundance of personal computers at commodity prices as well as the overwhelming publicity associated with the National Information Infrastructure is fostering a significant growth in the number of networked computer systems. While this growth is impressive, it is important to recognize that the value of networking to the end-user is limited by the degree to which the needed resources can actually be accessed after going *on-line*. The act of connecting a computer to a network, either physically or through wireless communication, does not guarantee that the user can access the resources available on the Internet. To a great extent, this access limitation is a direct result of the incompatibilities between different communications protocols.

As data communications evolved, many different protocols were developed to support new technologies and to address varying user requirements. This evolution has led to a diverse array of protocols that cannot interoperate. In order to promote interoperability, several standards organizations have worked to define standard protocols (e.g., TCP/IP, OSI). However, it is now clear that no single standard protocol or protocol family will become the universal protocol supported by all networked systems. Instead, large numbers of systems continue to be installed that support one of the standards or one of numerous proprietary protocols (e.g., IPX, AppleTalk, SNA). Even the Internet is no longer a single protocol network [15]. While TCP/IP remains the primary protocol suite, other protocols (e.g., IPX, AppleTalk, OSI) exist either natively or encapsulated as data within IP.

* 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.

Developing network systems that support multiple protocols can simplify the introduction of new protocols, like IPng, and reduce the risk for network managers faced with the prospect of supporting a new protocol. This will result in a faster, wider acceptance of new protocols and increased interoperability between network hosts. It has recently been pointed out that the National Information Infrastructure will be a multi-supplier, multi-technology endeavor that will create difficult interoperability problems. This will require mechanisms for negotiating commonality between network systems [32].

In our research, we consider ways in which multiprotocol networking can be accommodated through the use of multiprotocol systems [5]. We describe the need for *protocol discovery* in multiprotocol systems and present two mechanisms for performing this discovery. We also analyze the protocol features necessary to use multiprotocol systems and implement protocol discovery and we point out limitations in some current protocols. We also present some practical approaches to performing discovery and describe our experience in implementing multiprotocol systems.

Although multiple protocol support is of general importance, it can be specially applicable in certain wireless environments.

- Because of system resource limitations, wireless and mobile end-stations c an be rather inflexible in the protocols they support. For example, the Apple Newton will only support a specific flavor of AppleTalk protocols. This requires that a wired mobile support station support multiple protocols in order to be conversant with the different protocols that are being used by the mobile stations within range.
- Mobile end-stations can be made flexible by allowing them to download the protocols in use within a particular neighborhood. As the supported protocols can change from one neighborhood to another, this necessitates efficient protocol discovery services.

In the next section we present the background for this current work and a description of the multiprotocol model. We also discuss the need for protocol discovery in multiprotocol systems. In section 3 we analyze the use of directory services to assist in protocol discovery. This is followed in section 4 by the presentation of a different approach to discovery called protocol probing. We describe an integrated multiprotocol discovery system in section 5 and present our experiences in implementing multiprotocol systems. Related work is described in section 6. We conclude with a summary and future work in section 7.

2. Background

A simple definition of a multiprotocol system is "a host that supports more than one protocol or protocol family". Later in this section we will provide a more formal definition of protocols and multiprotocol systems but for now, this informal definition will suffice.

The data communications arena has witnessed the development of many different protocols. These protocols have been created over many years to support various new technologies and changing user requirements. Several protocols have evolved into standard protocol families (e.g., OSI, TCP/IP) while many others were developed as proprietary products (e.g., AppleTalk, DECNET, IPX, SNA). As a result of this diversity, network installations must support a wide variety of protocols to provide the connectivity demands of users.

Figure 1 depicts an example of the multiprotocol network environments we consider. This network consists of several hosts supporting one of several different protocol families. The interoperability of the systems is indicated by the dashed boxes. In this network an IP host can communicate with another IP host and an IPX host can likewise communicate with another IPX host. Unfortunately, even though the IP and IPX hosts are physically connected they cannot directly communicate without the use of a protocol translator or gateway [10,11].

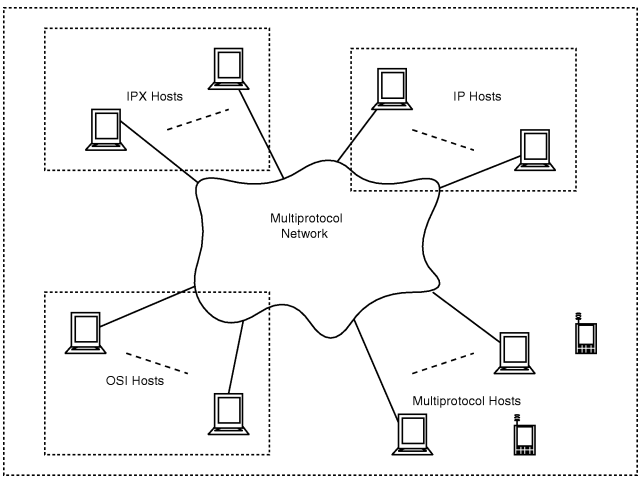


Figure 1. Multiprotocol network.

The multiprotocol hosts on this network are able to provide communication with hosts supporting a variety of protocols. Consider a multiprotocol system that supports IP, IPX, and OSI protocols. A user of this system would be able to perform communication functions (e.g., file transfer) with any of the other systems located on this network. In addition, the multiprotocol mobile systems can move freely from one protocol environment to another and still provide communication using the protocols that are supported in the new environment.

2.1. The multiprotocol model

The general problem we address is how to design network systems that can interoperate in a multiprotocol network. Our approach to achieving interoperability is to develop multiprotocol end-systems that can directly communicate with many different protocol configurations. In this section we present the model for our research.

In this work, we define a protocol as "a prior agreement among systems regarding the form and meaning of messages". A *protocol entity* (PE) is an object that implements a given protocol. With this protocol, a PE can communicate with another PE of the same type. In most cases, a PE of type a will use another PE of type b to transfer messages over to a's peer. This *uses* relationship between a and b gives rise to the common layering model that describes most network architectures. It is convenient to represent this relationship as a *protocol graph*. For instance, figure 2 portrays a multiprotocol protocol graph. In this graph, each box or node represents a PE and each edge represents a uses relationship. Each instance of communication invoked by a user of a protocol graph involves a particular subset of protocols in the graph. We refer to this subset as a *protocol path* or simply a *path*. A path encompasses a fixed set of PEs, connected by the *uses* relation, that provides communication from the top layer PEs down to the bottom layer PEs.

A system supporting the protocol graph in figure 2 provides a file-transfer service using nine different protocol

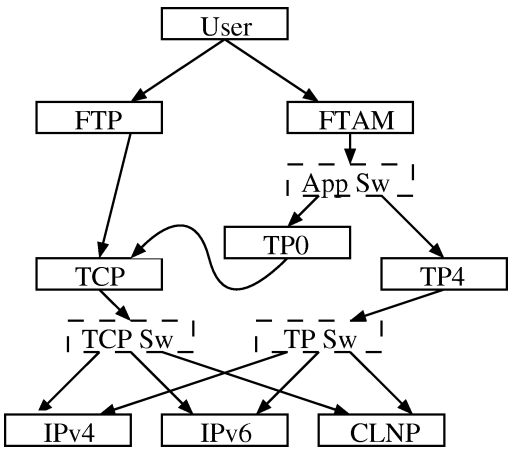


Figure 2. A multiprotocol graph.

paths. It supports the FTP application using TCP and the FTAM application using TP0 or TP4. This host supports three different network layer protocols. It supports the standard Internet protocol version 4, identified as IPv4. It also supports IPv6, the current proposed next generation Internet protocol (IPng) [12]. The OSI CLNP protocol is also supported along with the TUBA option [3] for providing TCP applications over CLNP.

It is likely that many future network systems will be configured to support multiple protocols, not unlike the system in figure 2. As new protocols are deployed, it is unreasonable to expect that users will be willing to give up any aspect of their current connectivity for the promise of a better future. For instance, most IPng installations will be made "in addition to" the current protocols. The resulting systems will resemble figure 2 in that they will be able to communicate with systems supporting several different protocols.

Unfortunately, in most current examples, multiprotocol architectures like that in figure 2 are implemented as independent protocol stacks running on a single system. This means, for instance, that even though both TCP and CLNP may exist on the system, there is no way to use TCP and CLNP in the same communication. The problem with such implementations is that they are designed as coexistence (or so-called "ships in the night") architectures and are not integrated interoperability systems. We believe future systems should include mechanisms to overcome this traditional limitation. By integrating the components of multiple protocol stacks in a systematic way, we can interoperate with hosts supporting any of the individual stacks as well as those supporting various combinations of the stacks.

In order to effectively use multiple protocols, a system must identify which of the available protocols to use for a given communication task. We call this the *Protocol Discovery* task. In performing this task, a system determines the combination of protocols necessary to provide the needed service. For achieving interoperability, protocols are selected from the intersection of those supported on the systems that must communicate. Two approaches to protocol discovery that we explore are the use of directory services to provide protocol configuration information and the use of feedback mechanisms in protocol probing. We present these two approaches separately in the next two sections. In section 5 we present a unified multiprotocol system that incorporates aspects of both approaches.

3. Protocol discovery using directory services

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 protocol discovery. Adding information about a host's protocols to this database is thus a rather natural way to support protocol discovery in a multiprotocol 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 information at all, because its protocol discovery problem is simple. To aid other multiprotocol 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 multiprotocol host have access to a distributed database that contains *some* encoding of the protocol information for the hosts with which it communicates.

3.1. An ideal directory service

In this section we present the design requirements for a directory service that most effectively supports the protocol discovery 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 implementations. Later we discuss how most of these features can be provided in the Internet Domain Name 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. An ideal multiprotocol directory service should, therefore, maintain 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.

A host's protocol graph information can be 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 indi-

Table 1
Protocol graph entries.

Multiprotocol		Single protocol	
PE name	Uses-list	PE name	Uses-list
FTAM	TP0 | TP4	FTP	TCP
FTP	TCP	TCP	IPv4
TP0	TCP	IPv4	
TP4	IPv4 | IPv6 | CLNP		
TCP	IPv4 | IPv6 | CLNP		
IPv4			
IPv6			
CLNP			

cates 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 multiprotocol graph shown in figure 2 and for the graph of a single protocol host. PEs with an empty uses-list are known as *base* PEs. These indicate that no lower layer matching information is available for protocol paths that include this PE. In general, the network layer protocols will serve as base PEs.

The two main functions of a directory service for multiprotocol 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 protocol 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 multiprotocol system. 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.

Each of the 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.

3.2. 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.

Our primary objective in this design is to develop a mechanism for delivering multiprotocol information that provides as many of the features identified in section 3.1 as possible while minimizing the impact on current directory service implementations. Our approach requires that additional DNS support be provided only in multiprotocol 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) [29]. 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 that could be deployed today.

3.3. An overview of DNS

The DNS, described in [20,21], 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 text field. The WKS record format 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 [27].

3.4. A multiprotocol 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 [20]. The most natural extension of DNS to the multiprotocol 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 multiprotocol 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 not pursued this approach since this change would conflict with our goal of not requiring the replacement of current systems.

We have identified several possible approaches to using the current DNS architecture for distributing multiprotocol host information. These approaches represent protocol configurations in the directory service by assigning an identifier to each PE. They all use the currently defined IN class resource records. In this paper we present an approach that uses the TXT RRs. This approach provides the most flexibility in encoding and presents the least danger of conflicting with current implementation and usage. Some alternative approaches we developed that use the WKS RR are detailed elsewhere [6].

```
<TXT Entry>   ::= "PEInfo:"<protos>
<protos>      ::= <protolist>";"<protos>
<protos>      ::= <protolist>
<protolist>   ::= <protolist>"|"<proto>
<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 3. A sample grammar for parsing PEInfo TXT entries.

```
;Name Class RR-Type RR-Data
;-----------------------------------------------
mphost  IN  A      127.1.1.1
mphost  IN  NSAP   49.5100bd5a00
mphost  IN  TXT    "PEInfo:IPv4;IPv6;CLNP"
mphost  IN  TXT    "PEInfo:TCP|TP4/IPv4|IPv6|CLNP"
mphost  IN  TXT    "PEInfo:TP0/TCP"
mphost  IN  TXT    "PEInfo:FTP/TCP"
mphost  IN  TXT    "PEInfo:FTAM/TP0|TP4"
```

Figure 4. A multiprotocol DNS entry.

3.5. Delivering protocol configurations using TXT resource records

We have designed and implemented a means for encoding protocol configurations using the DNS TXT entries. Currently, there is no widely used format for a TXT entry.[1] 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-list⟩/⟨uses-list⟩". The "|" symbol indicates disjunction. We use the leading string "PEInfo" to distinguish these protocol descriptions from other TXT fields in use. A grammar for parsing these TXT entries is presented in figure 3.

Figure 4 shows a possible DNS entry for a host with the protocol graph given in figure 2. This entry depicts the master file format used by the DNS server to store the domain information. Each line in this example corresponds to a separate RR. All of these RRs are associated with the

[1] Rosenbaum has proposed a new mechanism for using TXT fields for arbitrary string attributes [30]. At the time of this writing, this was an experimental Internet standard.

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 IPv4 address. The second line is an NSAP type record providing an OSI-format address. This format is described in [16] and is currently being updated [17].

The remaining lines contain TXT entries describing the host's supported protocols. 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 organization is strictly for convenience when maintaining the file. Note that we have provided explicit entries indicating the presence of the network layer protocols IPv4, IPv6 and CLNP. This is consistent with our goal outlined in section 3.1 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.

This protocol representation provides flexibility in encoding and does not conflict with current DNS implementation and usage. One issue to note is that while the TXT RRs are defined in the specification [21] 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 mentioned above [6] 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.

4. Protocol discovery using probing

While the directory service can provide useful information regarding protocol configurations, it is important to consider how a multiprotocol system should discover protocols when the directory service information is unavailable or inaccurate. This section describes an alternative protocol discovery technique called protocol probing. The idea behind protocol probing is to use the features of networks and the network protocols themselves in determining which protocol paths are available to support a given communication task. This is done primarily by attempting to communicate using different protocols and monitoring the attempts to see what can be learned about the network configuration. In protocol probing, we use the information gained from these attempts to determine which protocols are supported on the remote system.

Here we present the protocol probing concept, beginning in the next section with a simple example. After the example we give a formal description of the probing task and present protocol features necessary for probing.

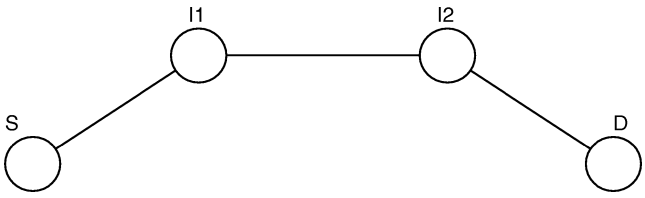


Figure 5. A sample network.

4.1. An example of protocol probing

Consider an initiating host supporting a multiprotocol graph such as that of figure 2. This is the source or initiating host, labeled *S*, in the network with topology presented in figure 5. The user of this host is attempting to communicate with the destination or responding host *D*. These hosts are physically connected via two intermediate network layer routers labeled $I1$ and $I2$.

In order to perform a file transfer, the user must first determine which, if any, of the nine distinct local protocol combinations are supported on the remote host. The task is to determine which protocols supported by S are also supported by D. It is also necessary to determine the network layer protocols supported by both $I1$ and $I2$ as well as S and D.

In this case, without prior knowledge of the remote configuration, the user determines the protocols to use based on the feedback provided from the applications. The user proceeds by attempting a connection with one of the two applications and monitoring the way this attempt fails or succeeds. A successful connection indicates that the current application could be used. A failed connection indicates one of several possible problems.

Suppose for instance that host *D* supports the single OSI stack consisting of FTAM/TP4/CLNP. If the user decides to first try the FTP application with TCP and IP then this attempt will subsequently fail.[2] Table 2 lists the feedback provided to the user for various protocol incompatibilities.[3] In this case, since no compatible network layer is found, the user will receive the *Connection timed out* message. Since this message does not provide information about the actual cause of the failure, there is little to assist the user in choosing the next protocol combination to try. If the user continues by trying FTP with the two other options, both will fail. However, the attempt with the FTP/TCP/CLNP combination will fail with the *Connection refused* message. Based on the information in table 2, the user can determine that CLNP is supported on the remote system. Now, when the FTAM application is tried, the user can choose the protocol combination that includes CLNP. In this example, the FTAM/TP4/CLNP option is tried and the communication succeeds.

[2] The first type of failure that might be encountered is the failure to find a network address for host D in the format required for the selected protocol. In order to simplify the example, the discussion here assumes that addresses are available for all the protocols attempted.

[3] The error messages listed are for the FTP program from Sun OS 4.1.3 and FTAM program from the ISODE version 8.

Table 2
FTP and FTAM error messages.

Compatibility problem	FTP error message	FTAM error message
Network congestion/partition	Connection timed out	Timer expired
Remote host off-line	Connection timed out	Timer expired
No compatible physical layer	Connection timed out	Timer expired
No compatible network layer	Connection timed out	Timer expired
No compatible transport layer	Connection refused	Timer expired
No compatible application layer	Connection refused	OSI service tsap#259 not found

4.2. The protocol probing task

The "guided" protocol selection carried out in the above scenario is what we are interested in with protocol probing. At the beginning of performing protocol probing the paths currently supported on the multiprotocol system S are known. Given this, it is necessary to find a path that is supported on both S, D, and the intermediate systems $I1$ and $I2$. In protocol probing, this determination is made based on feedback from interaction with the network, primarily through communication attempts with various protocols. Until negative feedback arrives about a protocol in a path, it may still be the case that the path is supported on D. In general, it can only be determined that a path is not supported on D after feedback arrives from some protocol indicating that the path, or some protocol in the path, is not supported on D. Because of intermittent failures and delays, it is not possible to assume a path is not supported on D based only on the failure of an attempt to use the path. On the other hand, positive feedback about a path on D may come as a result of successful interaction with all of the protocols in the path or from another protocol indicating that the path is supported on D.

In protocol discovery, using either directory services or probing, it is not necessary to determine the set of all paths supported by D. All that is needed is to find one common path between D and S that uses the same protocols and will therefore provide sufficient communication. The probing process performs a bottom-up search of the protocol graph to find a matching protocol path. The probing system starts with a low level protocol, like the CLNP network layer, and tries to establish whether it exists on the destination. Once positive feedback arrives from the network layer, the next step is to determine which transport layer protocols are available through this network layer. Eventually, when communication is established with some application that provides the service requested, the probing process can stop and normal communication can proceed. In this case we say that a complete, matching protocol path has been found. It is possible that an initially successful communication attempt will not lead to complete success. For instance, even after identifying a network and transport layer, it may still be necessary to backtrack and find another network layer if the first one does not lead to an application providing the desired service.

4.3. Issues with protocol probing

In order to use protocol probing it is necessary to develop an algorithm or set of rules that directs the operation of the probing process. In this section we present several factors that affect the design of these algorithms.

- *Where to start:* The first interesting question in designing a probing algorithm is, "Which protocol should be tried first?" The directory service described in section 3 should provide a good indication of protocols to start with. A mobile multiprotocol system, when moved to a new location, might listen for other protocol traffic on the new network. If most of the other hosts appear to support one specific protocol family, then those protocols should probably be tried first.
  Another good choice for a first try is a protocol that provides particularly good feedback when failures occur. In this case, even if the protocol does not establish the desired communication, it should provide good insight into which protocols to try next.
- *Interpreting failure:* The most important aspect of the probing process is the use of failed communication attempts to learn as much as possible about the remote configuration. The information learned is then used to guide the selection of the next protocol to be attempted. The extent to which this is possible depends on the type of feedback provided by a protocol when it fails. In section 4.1 we presented an example where more detailed feedback would be useful for user probing. In the next section we give more detail on the type of feedback needed for probing.
- *When to give up:* Many failed attempts will result in no feedback at all. In these cases it is difficult to distinguish a situation of temporary network congestion or failure from a case where the protocols being used are not supported on the remote system. The first decision of this sort is to decide when to stop trying one protocol and move on to another. Additionally, the probing algorithm may be designed to start over after unsuccessfully trying all the protocol combinations and try them again. In this case the system must decide when to abandon this probing cycle.
- *Datagram vs. connection service:* Probing fits most naturally as part of the connection establishment process. This is because of the fact that positive feedback of communication success is provided as part of connec-

tion establishment. Datagram service is often unreliable and a "no feedback" situation could mean that the communication was a success. In this case, higher level indication from the application or user is necessary to provide enough context to indicate when a communication attempt was successful.

- *Access to protocol operation:* The feedback mechanism supported by a protocol is only useful for protocol probing if the probing system has access to it during each communication attempt. In many cases, protocol implementations do not make the complete feedback information available to the protocol users. It may also be useful for the probing system to have access into the operation of the protocol. One example of this is a system that uses TCP as one of the possible transport protocols. While the TCP handshake is taking place, some feedback has been received but the upper layer has not yet heard indication of a connection. If the probing system is about to give up on this attempt, it could check on the state of the TCP protocol and detect that enough progress has been made to warrant waiting further.
  In order to provide this type of access to protocol operations it is necessary to "open up" the details of the actual protocol implementations. This will be difficult for applications implemented as user level programs. Such programs will have protection problems in getting to detailed protocol information in popular systems like UNIX where the protocols are commonly implemented in a privileged kernel space.
- *Parallel attempts:* The probing algorithm can be designed to try protocols one at a time or it could try several protocols in parallel. In the parallel case, all the possible protocol combinations would be tried at the same time and the first one to successfully communicate would be handed off to the user. The parallel approach might be practical when parallel hardware is available or when there is a long propagation delay between the two systems and trying several protocols is faster than waiting for feedback.
  One problem with this approach is that it may result in the creation of several successfully communicating sessions, all but one of which would need to be gracefully terminated without being used.

4.4. Feedback

The feedback provided by protocols regarding communication failures is important enough to the probing process to warrant further discussion. As we described in section 4.2, the only way a probing system S can determine for sure whether a protocol path is supported on D is to obtain some definite feedback indication.

The most effective way to obtain positive feedback about the existence of a path is to get it directly from the protocols that constitute the path. This feedback could be in the form of a direct acknowledgment of transmitted data or it could be any other data received from the destination that uses the protocols in the path. This second approach is necessary when doing probing with unreliable datagram protocols that do not send back acknowledgments. It is also possible to get feedback from other protocols, which are not part of a path, indicating that the path is supported on D. While such feedback would strongly suggest that the path is supported, the only sure way to guarantee that a path is truly available for communication is to hear it directly from the protocols in the path.

Unlike positive feedback, negative feedback indicating that a path is not supported on D will, in general, need to be sent by some protocol other than the one missing from D. Most of the time, useful negative feedback will come from some lower layer protocol that carries the traffic of the missing protocol. For instance, a network layer protocol might return an error indicating that the desired transport layer protocol was not found. The only example we have encountered where negative feedback might come from the protocol itself is the case of network protocols that send feedback from an intermediate router (e.g., $I1$ in figure 5) indicating that D is not reachable using this network protocol.

Clearly it is impractical for D to provide complete feedback for every protocol message sent by S; in the case of an unreliable connection-less protocol, this would conflict with the design intention of the protocol. One option that may be useful for such protocols is to provide a variable feedback mechanism that allows the sender to request positive feedback on certain data. This feedback could be turned on by S for the first several datagrams while probing is taking place. After the protocols are determined and communication succeeds, the feedback could be turned off to provide the efficiency normally desired for such protocols.

We propose that future protocols include a three level feedback mechanism. The feedback level is indicated by the value of a 2-bit field in the protocol header. This field tells the end and intermediate systems what type of feedback should be sent to S about the processing of this packet. The highest feedback level indicates both positive and negative feedback. This is the level used during protocol probing. It allows both end and intermediate systems to provide an immediate indication of the degree of success achieved with this protocol. After determining which protocols to use, the source sets the feedback field of all subsequent outgoing packets to request only negative feedback. This is the normal case for most feedback mechanisms today. The third feedback level indicates that no feedback should be provided to S regarding this packet. This option is useful for any application where feedback is unnecessary or could overwhelm the source.

4.5. Feedback analysis

This section provides a detailed analysis of the feedback mechanisms available in IP and CLNP that could be used to support probing. With IP, network layer feedback is provided by the Internet Control Message Protocol

Table 3
Protocol problem feedback for IPv4.

Compatibility problem	ICMP feedback	Generated by
No IPv4 at $I1$	None, timeout	S
No IPv4 at $I2$	Net unreachable	$I1$
No IPv4 at D	Host unreachable	$I2$
No matching transport at D	Protocol unreachable	D
No matching application at D	Port unreachable	D
IP option mismatch at $I1$, $I2$, D	Parameter problem	$I1$, $I2$, D
Time exceeded at $I1$, $I2$	Time exceeded in transit	$I1$, $I2$
Time exceeded at D	Time exceeded in reassembly	D

Table 4
Protocol problem feedback for CLNP.

Compatibility problem	CLNP feedback	Generated by
No CLNP at $I1$	None, timeout	S
No CLNP at $I2$	Destination unreachable	$I1$
No CLNP at D	Destination unknown	$I2$
No matching transport at D	None	D
No matching application at D	None	D
CLNP option mismatch at $I1$, $I2$, D	Unsupported option	$I1$, $I2$, D
Time exceeded at $I1$, $I2$	Lifetime expired in transit	$I1$, $I2$
Time exceeded at D	Reassembly lifetime expired	D

(ICMP) [25]. ICMP is an unusual protocol in that it is both an integral part of IP and a user of IP, using IP to transfer its messages. ICMP has several different message types, most of which are used to provide feedback during communication. These messages are generated by a network host when it encounters an error while processing an IP datagram.

Table 3 gives the ICMP feedback that would be generated for several possible protocol compatibility problems. The most difficult problem to recognize is when there is no compatible network layer at $I1$, the next hop from the source. When this occurs, there is no direct feedback and therefore a failure can only be inferred by a timeout in a higher layer protocol. For the other compatibility problems listed in table 3 there are unique feedback messages in ICMP to indicate the problem. These messages can be used by a multiprotocol system to determine where an incompatibility occurs. The feedback presently proposed for IPv6 is a straightforward extension of the current ICMP with few differences [9].

Unlike IPv4 and IPv6, CLNP includes a feedback mechanism as part of the network protocol definition. CLNP supports a number of *Error Report (ER) PDUs* that provide feedback during the operation of the network layer protocol. Most of the ER messages are designed to provide specific feedback for the operation of CLNP itself.

The ER PDUs returned for various protocol compatibility problems are given in table 4. Two important messages that are not provided by the CLNP ER PDUs are the Protocol and Port Unreachable messages found in ICMP. These messages are particularly useful in multiprotocol networks where there may be several different transport and application layer protocols. Like ICMP, there is no means for CLNP to automatically detect that the protocol is not supported on the next hop $I1$. A timeout mechanism is also required in this case to detect such a failure. CLNP provides a two level feedback mechanism that allows the source to specify whether or not negative feedback should be returned. Feedback is requested by setting the error report (ER) flag in the header of each packet for which feedback is desired.

4.6. A protocol probing algorithm

Based on the feedback analysis of the previous section, we developed algorithms for carrying out protocol probing with the protocols studied. An example of such a probing algorithm is presented in figure 6. This algorithm begins in the upper left block with the TCP/IPv4 protocols. We chose this as our starting point since a large percentage of the systems on our network support these protocols. If the attempt succeeds then the connection is established and communication proceeds. If the attempt fails, this algorithm takes one of three different paths depending on the feedback given. If an ICMP *Protocol Unreachable* is received then we know that IPv4 is supported on the remote system and it is probably best to try a different upper layer protocol. If a TCP *Reset* message is received then we know that both IPv4 and TCP are supported but the requested application was inaccessible through TCP. The system could still support this application over TCP using the RFC-1006 mechanism for providing OSI applications over TCP. If TCP is found, the RFC-1006 option is attempted next. If TCP is not found but IPv4 is, then the TP4 protocol is attempted with IPv4. If neither of these feedback messages are received, the system assumes that IPv4 is not supported on the remote system and goes on to try the TCP/IPv6 combination.

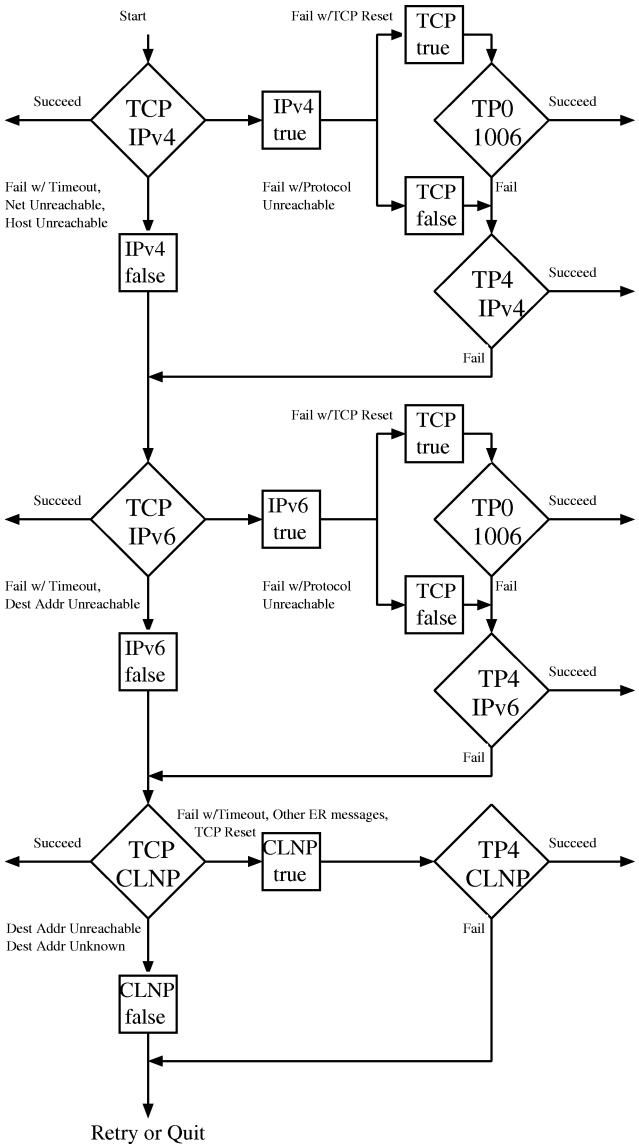


Figure 6. Protocol probing algorithm.

The IPv6 portion of the algorithm proceeds much like the IPv4 portion since the ICMP feedback is similar for the two protocols. In our implementation, discussed in section 5.2, we do not yet support the IPv6 protocol so this portion of the algorithm is not used. The CLNP algorithm does not include the two options for higher level protocol problems since it does not include specific feedback about unreachables.[4] For CLNP, if a packet arrives for an unsupported upper layer protocol the packet is simply discarded. This means that the case of no feedback from a CLNP attempt could still mean that CLNP is supported on the system. When TP4 is found on the remote system but the application is not available, TP4 will return a *Disconnect Request* TPDU. In the algorithm presented here, this feedback does not affect the order of protocol attempts.

[4] The TUBA proposal [3] recommends the addition of these as two new ER types.

5. An integrated protocol discovery architecture

While it is possible to develop practical multiprotocol systems that use only one of directory services or probing to perform protocol discovery, we feel it is best to incorporate aspects of both approaches in multiprotocol systems. In this section we describe such an integrated system and present issues uncovered in our implementation experience.

We have identified the following two basic approaches to supporting protocol discovery in multiprotocol systems.

- *User based discovery.* In this approach, the user performs discovery by querying the directory service and then attempting to use different protocol paths. This is the approach carried out in the previous example using FTP and FTAM. To support this approach it is necessary to enhance the current protocol systems to provide explicit failure indications. In the above scenario, a message such as *CLNP supported, TP4 not supported* rather than the simple *Connection refused* would greatly assist the user in determining which protocol path to attempt next.
- *Automated protocol discovery system.* This approach involves implementing a protocol subsystem that performs protocol discovery automatically as part of the normal protocol operations. Such a system could receive a communication request from the user such as, “transfer files from host A to host B”, and then determine which protocols to use to perform the task.

In the current work, we focus on the second approach. A goal of this research is to develop the necessary components for a multiprotocol system that can take a user’s request and return a “connected” communication session. This system will operate without the user being aware of the protocols used, directory services accessed, different network address formats, or failures during protocol attempts. In this system, feedback is given to the user only after the communication is successfully established or all possible protocol combinations are exhausted.

5.1. Automated discovery architectures

Here we describe three different architectures for developing an automated discovery system. Each architecture has its own virtues and limitations. As we will see, the main distinction among the architectures is the scope of the discovery system in terms of the protocols included. This difference in scope is depicted in figure 7.

The first approach to automated discovery is to perform protocol discovery as part of a *Generic Application* that provides a common service. The generic File Transfer Service of figure 7 presents the user with a consistent interface, regardless of the actual protocols or applications used [28]. This approach incorporates the entire protocol graph, including the applications, into the discovery system. The main advantage of this approach is that it allows the user to communicate with a wide range of currently-installed

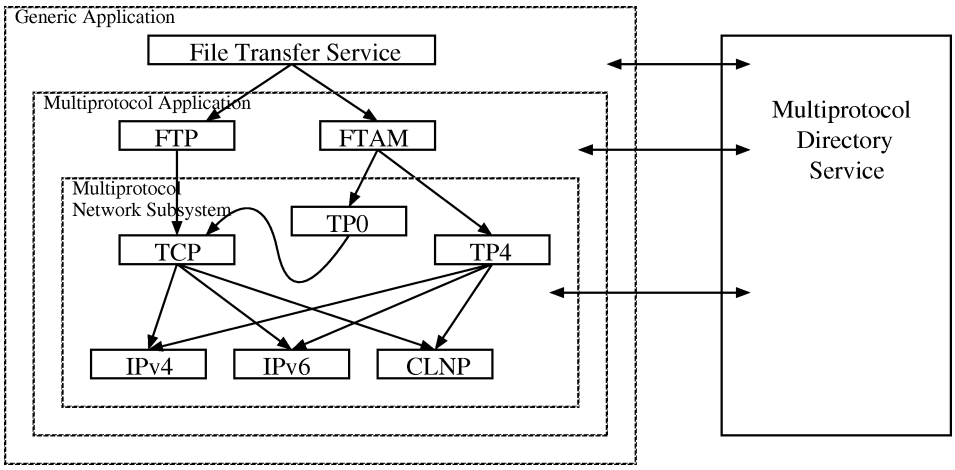


Figure 7. Automated discovery architectures.

systems, including those supporting a variety of applications.

A problem with this approach is that it is difficult to hide the actual applications and some limitations will always arise when developing the generic interface. For instance, the FTAM application includes several features not found in FTP. These features will need to be either emulated for FTP communications or not provided at all.

The second approach to performing automated protocol discovery is to develop *Multiprotocol Applications* that perform discovery themselves. For example, a multiprotocol FTP implementation could support discovery by including calls to several different protocols.[5] This approach provides the user with a familiar user interface and functionality while hiding many of the details of protocol discovery. The user will still need to perform part of the discovery process by selecting the application that is supported on the remote host.

One disadvantage of this approach is that it only provides communication with systems that support some version of this multiprotocol application. In the short term, this application will primarily be supported only over its native protocols (e.g., FTP over TCP/IP). This will limit the connectivity attainable by the multiprotocol system. Another drawback is that this approach may require extensive modification of the application to support new protocols and address formats.

The effort needed to modify the application to support multiple protocols can be alleviated by performing protocol discovery below the application/protocol interface in a *Multiprotocol Network Subsystem*. This is the idea behind our third approach where a multiprotocol network subsystem is used to provide multiprotocol support through standard programming interfaces. The network subsystem is the portion of a host operating system that supports protocol implementations. Two popular examples are the System V Streams [2] and BSD UNIX Socket [14] environments. Implementing protocol discovery as part of the network subsystem enables current applications to run over multiple protocols with little or no modification to the actual application. The degree of connectivity provided is essentially the same as in the multiprotocol application approach.

The main drawback of this approach is that it requires extensive modification of the network subsystem. Also, as with the generic application, the use of a generic interface to many protocols will usually result in a compromise of functionality for some of the protocols. For instance, TCP provides a graceful disconnect while the OSI TP4 does not. Another issue is that since the discovery process is performed automatically, the application programmer loses some control over the actual protocols used as well as the discovery process itself.

5.2. Discovery system implementation

To demonstrate the feasibility of the protocol discovery concept we developed an implementation incorporating two of the three network protocols from figure 2: IPv4 and CLNP.[6] We pursued both the multiprotocol application and multiprotocol network subsystem architectural approaches (see section 5.1). This implementation was done in Sun OS 4.1.3 for the Sun SPARC architecture. This work involved the addition of the CLNP protocol[7] as well as the development of the discovery system we describe.

For the directory services implementation we developed a set of extensions to the BSD DNS resolver library. We added the *LookupHost*( ) and *MatchPath*( ) functions discussed in section 3.1. 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 a list of paths

[5] Some of the changes needed to provide this support in FTP have already been proposed [24].

[6] The IPv6 protocol is still evolving. We have been closely following the standardization effort and will be incorporating IPv6 into our implementation as the specification matures.

[7] For the CLNP portion we made significant use of code from the NetBSD software release. Additionally, we are indebted to Francis Dupont for the contributions made to support TUBA.

that support the required service. The discovery system implementation utilizes the probing algorithm presented in figure 6. This algorithm performs probing of protocols that provide a connection-oriented service.

In our implementation of the multiprotocol application architecture we perform discovery as part of a multiprotocol FTAM application. The application calls the *LookupHost*( ) and *MatchPath*( ) DNS routines to determine the likely protocols and addresses to use. For each protocol combination attempted by the algorithm, the application creates a new socket using the appropriate protocols. The standard socket interface does not provide the level of feedback required by our probing algorithm. In order to provide feedback regarding the actual failures we added several new error codes (i.e., errno values) that indicate the ICMP return codes.

To implement the multiprotocol network subsystem, we incorporated discovery into the BSD socket architecture. In this implementation we introduced the novel concept of a *multiprotocol family*. This new protocol family is denoted as PF_MULTI. To use the multiprotocol system, a programmer creates a socket with this protocol family and the protocol service type required (e.g., datagram or stream). When the socket is created it is still not known exactly which protocols will be used to implement this socket. The discovery system is invoked when the user attempts to establish a connection via the *connect*( ) system call. After the protocols are determined, the protocol specific values are filled in to the socket structure. For datagram service, protocol discovery is performed at the time of the first data send (e.g., with the *sendto*( ) system call).

It is important to note that our multiprotocol subsystem version of automated discovery is implemented within the UNIX kernel and has access to the entire set of protocol implementations and the feedback they provide. When implementing an automated discovery system in other architectures where protocols are not part of the same privileged address space (e.g., Mach [1]), it will be important to provide access to the protocol feedback systems.

One interesting challenge with this architecture is how to specify the appropriate address information for each of the protocols that will be attempted. In current systems, the application creates an address structure of the appropriate type (e.g., AF_INET) and passes it in as an argument to the *connect*( ) system call. With our system, it is necessary to have addresses for each of the several different protocols that will be attempted. The application calls the DNS resolver functions and then passes the results on to the subsystem which carries out the protocol probing. We provide these in the *connect*( ) call as a linked list. We felt this to be a better approach than the alternative of having the PF_MULTI domain code call the directory service directly.

Both of our implementations provide protocol discovery that enables the user of an application on a multiprotocol system to communicate with hosts supporting any of five different protocol combinations. The protocol subsystem approach provides this support without requiring the application programmer to implement discovery.

6. Related work

In most cases, what is meant by multiprotocol support is that a system includes multiple independent protocol environments; the so called “ships in the night” approach popular with multiprotocol routers. This is similar to the “dual-stack” architecture described in [28] where a single system supports two independent protocol stacks with no communication between them.

Encapsulation or tunneling is commonly used where two networks that support a common protocol must be connected using a third intermediate network running a different protocol. This approach is only appropriate when both communicating end systems support the same protocol stack. It does not provide interoperability between these end systems and systems running the protocol supported in the intermediate network. Some examples of this approach are: a mechanism for providing the OSI transport services on top of the Internet protocols [4], encapsulating IEEE 802.2 frames in IPX network packets [18], tunneling IPX [26] and AppleTalk traffic over the Internet backbone.

A great deal of previous research has focused on addressing the problem of multiple protocols through the use of translation or conversion gateways. In [10], Green describes the conversion task and characterizes the aspects of communications protocols that must be considered when defining a protocol converter. Several examples of protocol converters exist in the literature, most of them residing at the application layer [11,28,29].

Recently, others have begun to research the issues involved in integrating protocols from different architectures. Ogle et al. [23] are developing a TCP/IP and SNA system that performs protocol selection below the socket level interface. Janson et al. [13] consider options for interoperability between OSI and SNA networks, and analyze the addressing issues arising when these protocols are combined in a single network.

The diversity of communicating systems was addressed in the Heterogeneous Computer Systems (HCS) Project at the University of Washington [22]. They develop a Remote Procedure Call (RPC) based environment for developing implementations that are independent of underlying protocol systems. Another approach to achieving interoperability in diverse systems is to provide a mechanism for hosts to exchange protocol information before carrying out the communication task. Two early examples of this approach are the Network Command Language described by Falcone [8] and the “meta-protocol” concept proposed by Meandzija [19]. Similarly, Tschudin describes a “generic protocol” in [31].

The recent work of Comer and Lin [7] describes the use of a technique called *active probing* to deduce characteristics of a TCP implementation. While their work focuses on discovering possible problems with a known protocol, the technique used is somewhat similar to the probing process we perform.

7. Concluding remarks

Conventional wisdom once held that a single standard protocol architecture would eventually "win out" over all others, thus guaranteeing interoperability among all systems. In contrast, today's communications environment is one of multiple, co-existing protocols. This paper describes the development of systems that support several different protocols. These multiprotocol systems are likely to be significant in providing interoperability in heterogeneous environments. Multiprotocol systems may also be practical in the wireless mobile computing world where systems move between different protocol environments.

A challenge for multiprotocol systems is to determine which of the several supported protocols should be used for a given communication task. We have developed two approaches to performing this protocol discovery. The first approach uses directory services to deliver protocol configuration for a system. For this approach, we have described protocol representations and an implementation solution that can be deployed today with currently available Internet software. The second protocol discovery approach presented is protocol probing. This technique uses the feedback from communication attempts to determine which protocols are currently supported. We present a detailed analysis of current feedback mechanisms and describe features that would be useful for future protocols.

In this work we describe three main architectures for developing multiprotocol systems that automatically perform discovery. These architectures offer varying levels of interoperability with other network systems. While the generic application offers the user seamless connectivity to the widest range of systems, this approach has the highest development cost since much of the implementation can only be used for the specific application it was developed for. The multiprotocol subsystem approach allows a single discovery implementation to support several different applications.

The probing algorithm presented here was designed after analyzing the feedback provided by the protocols in use, studying the implementation options in our development environment, and using empirical evidence to decide which protocols were the more likely to succeed after each failure. Our future work will include the continued study of discovery algorithms for different protocols and the exploration of ways to simplify the design of probing algorithms. We will also look at the area of protocol subsystem requirements for multiprotocol systems that can rapidly change protocol configurations through mechanisms such as protocol downloading.

References

[1] M. Accetta, R. Baron, W. Bolosky, D. Golub, R. Rashid, A. Tevanian and M. Young, Mach: A new kernel foundation for Unix development, in: *Proceedings Summer Usenix* (July 1986).

[2] AT&T, *Unix System V, STREAMS Programmer's Guide* (1988).

[3] R. Callon, TCP and UDP with bigger addresses (TUBA), a simple proposal for internet addressing and routing, Request for Comments (Informational) RFC 1347, Internet Engineering Task Force (June 1992).

[4] D. Cass and M. Rose, ISO transport services on top of the TCP: version 3, Request for Comments (Standard) RFC 1006, Internet Engineering Task Force (May 1987). Obsoletes RFC 983.

[5] 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* (IEEE, April 1993) pp. 136–143.

[6] R.J. Clark, K.L. Calvert and M.H. Ammar, On the use of directory services to support multiprotocol interoperability, in: *Proceedings of IEEE INFOCOM* (IEEE, June 1994) pp. 784–791.

[7] D.E. Comer and J.C. Lin, Probing TCP implementations, in: *Summer USENIX* (June 1994) pp. 245–255.

[8] J.R. Falcone, A programmable interface language for heterogeneous distributed systems, ACM Transactions on Computer Systems 5(4) (November 1987) 330–351.

[9] R. Govindan and S. Deering, ICMP and IGMP for the simple internet protocol plus (SIPP), Internet Draft (March 1994).

[10] P.E. Green, Protocol conversion, IEEE Transactions on Communications 34(3) (March 1986) 257–268.

[11] I. Groenbaek, Conversion between the TCP and transport protocols as a method for achieving interoperability between data communications systems, IEEE Journal on Selected Areas in Communications 4 (February 1986).

[12] R. Hinden, Internet protocol, version 6 (IPv6) specification, Internet Draft (October 1994).

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

[14] S.J. Leffler, M.K. McKusick, M.J. Karels and J.S. Quarterman, *The Design and Implementation of the 4.3 BSD UNIX Operating System* (Addison-Wesley, Reading, MA, 1989).

[15] B. Leiner and Y. Rekhter, The multiProtocol Internet, Request for Comments (Informational) RFC 1560, Internet Engineering Task Force (December 1993).

[16] B. Manning, DNS NSAP RRs, Request for Comments (Proposed Standard) RFC 1348, Internet Engineering Task Force (July 1992). Obsoleted by RFC1637.

[17] B. Manning and R. Colella, DNS NSAP resource records, Internet Draft (December 1993).

[18] L. McLaughlin, Standard for the transmission of 802.2 packets over IPX networks, Request for Comments (Standard) RFC 1132, Internet Engineering Task Force (November 1989).

[19] B. Meandzija, Integration through meta-communication, in: *Proceedings of IEEE INFOCOM* (June 1990) pp. 702–709.

[20] P. Mockapetris, Domain names – concepts and facilities, Request for Comments (Standard) RFC 1034, Internet Engineering Task Force (November 1987). Obsoletes RFC 973; updated by RFC1101.

[21] P. Mockapetris, Domain names – implementation and specification, Request for Comments (Standard) RFC 1035, Internet Engineering Task Force (November 1987). Obsoletes RFC 973; updated by RFC1348.

[22] D. Notkin, A.P. Black, E.D. Lazowska, H.M. Levy, J. Sanislo and J. Zahorjan, Interconnecting heterogeneous computer systems, Communications of the ACM 31(3) (March 1988) 258–273.

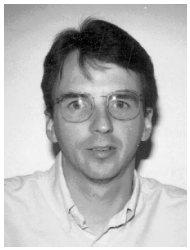
[23] D.M. Ogle, K.M. Tracey, R.A. Floyd and G. Bollella, Dynamically selecting protocols for socket applications, IEEE Network 7(3) (May 1993) 48–57.

[24] D. Piscitello, FTP operation over big address records (FOOBAR), Request for Comments (Proposed Standard) RFC 1639, Internet Engineering Task Force (June 1994). Obsoletes RFC1545.

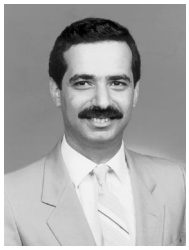
[25] J. Postel, Internet control message protocol, Request for Comments (Standard) RFC 792, Internet Engineering Task Force (September 1981). Obsoletes RFC 777.

[26] D. Provan, Tunneling IPX traffic through IP networks, Request for Comments (Experimental) RFC 1234, Internet Engineering Task

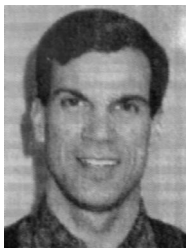
Force (June 1991).
[27] J. Reynolds and J. Postel, Assigned numbers, Request for Comments (Experimental) RFC 1340, Internet Engineering Task Force (July 1992). Obsoletes RFC1060; Obsoleted by RFC1700.
[28] M.T. Rose, *The Open Book* (Prentice-Hall, Englewood Cliffs, NJ, 1990).
[29] M.T. Rose, *The ISO Development Environment User's Manual – Version 7.0* (Performance Systems International, July 1991).
[30] R. Rosenbaum, Using the domain name system to store arbitrary string attributes, Request for Comments (Proposed Standard) RFC 1464, Internet Engineering Task Force (May 1993).
[31] C. Tschudin, Flexible protocol stacks, in: *Computer Communication Review* (ACM Press, September 1991) pp. 197–205.
[32] M.K. Vernon, E.D. Lazowska and S.D. Personick, *R&D for the NII: Technical Challenges* (Interuniversity Communications Council, Inc. (EDUCOM), 1994).



Russell Clark holds the B.S. (1987) in mathematics and computer science from Vanderbilt University and the M.S. (1992) and Ph.D. (1995) in information and computer science from the Georgia Institute of Technology. Dr. Clark is currently a Senior Scientist with Empire Technologies, Inc. He was an Assistant Professor at the University of Dayton in Dayton, Ohio, from 1995 until 1997. He was a Software Engineer at Data General Corporation for the years 1987–1990. His research interests include interoperable systems using the use of multiple protocols, scalable network services using multicast distribution, and automated network management systems. Dr. Clark is a member of the ACM and a member of the IEEE Computer Society.
E-mail: rjc@empiretech.com



Mostafa H. Ammar received his Ph.D. degree in electrical engineering from the University of Waterloo in Ontario, Canada, 1985. His S.M. (1980), S.B. (1978) degrees were acquired in electrical engineering and computer science at the Massachusetts Institute of Technology, Cambridge, MA. Dr. Ammar is currently an Associate Professor in the College of Computing at Georgia Tech. He has been with Georgia Tech since 1985. For the years 1980–1982 he worked at Bell-Northern Research (BNR), in Ottawa, Ontario, Canada, first as a Member of Technical Staff and then as Manager of Data Network Planning. Dr. Ammar's research interests are in the areas of computer network architectures and protocols, multipoint communication, distributed computing systems, and performance evaluation. He is the co-author of the textbook "Fundamentals of Telecommunication Networks", published by John Wiley and Sons. Dr. Ammar is the holder of a 1990–1991 Lilly Teaching Fellowship and received the 1993 Outstanding Faculty Research Award from the College of Computing. He is a member of the editorial board of IEEE/ACM Transactions on Networking and Computer Networks and ISDN Systems Journal. He is also the co-guest editor of a recent issue (April 1997) of IEEE Journal on Selected Areas in Communication on "Network Support for Multipoint Communication". He is the Technical Program Co-Chair for the 1997 IEEE International Conference on Network Protocols. Dr. Ammar is a Senior Member of the IEEE and a member of the ACM and a member of the Association of Professional Engineers of the Province of Ontario, Canada.
E-mail: ammar@cc.gatech.edu



Kenneth L. Calvert received the Ph.D. degree in 1991 from the Department of Computer Sciences at the University of Texas at Austin. Since 1991 he has been an Assistant Professor in the College of Computing at the Georgia Institute of Technology, where his research deals with the design and implementation of high-performance communication protocols and services, with an emphasis on reducing the cost of deploying application-specific services. Additional research interests include network security, topological models of the Internet, and the use of formal methods to manage complexity. Dr. Calvert received the 1996 "gus" Baird Teaching Award and the 1996 E-Systems Faculty Fellowship Award from the College of Computing at Georgia Tech. From 1979 to 1984 he was a Member of the Technical Staff at Bell Laboratories in Holmdel, New Jersey. He is a member of the ACM and the IEEE.
E-mail: calvert@cc.gatech.edu