June 1988 Report No. STAN-CS-88-1213

(Also numbered CSL-TR-88-360)

2

Exploiting Recursion to Simplify RPC Communication Architectures

by

David R. Cheriton

Department of Computer Science

Stanford University
Stanford, California 94305

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188
Exp Date Jun 30, 1986

1a REPORT SECURITY CLASSIFICATION	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) STAN-CS-88-1213	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Computer Science Dept. 6b OFFICE SYMBOL (If applicable)	7a NAME OF MONITORING ORGANIZATION
6c ADDRESS (City, State, and ZIP Code) Stanford University Stanford, CA 94305	7b ADDRESS (City, State, and ZIP Code)
8a NAME OF FUNDING/SPONSORING ORGANIZATION DARPA 8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO · PROJECT NO · TASK NO · WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification)

Exploiting Recursion to Simplify RPC Communication Architectures

12 PERSONAL AUTHOR(S) David R. Cheriton

13a TYPE OF REPORT	13b TIME COVERED FROM ________ TO ________	14 DATE OF REPORT (Year, Month, Day) June 1988	15 PAGE COUNT 12

16 SUPPLEMENTARY NOTATION

17 COSATI CODES		
FIELD	GROUP	SUB-GROUP

18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

Current communication architectures suffer from a growing collection of protocols in the host operating systems, gateways and applications, resulting in increasing implementation and maintenance cost, unreliability and difficulties with interoperability. The *remote procedure call (RPC)* approach has been used in some distributed systems to contain the diversity of application layer protocols within the procedure call abstraction. However, the same technique cannot be applied to lower layer protocols without violating the strict notion of layers.

In this paper, we show how the RPC approach can be used for lower layer protocols so that the resulting "layer violations" generate a simple recursive structure. The benefits of exploiting recursion in a communication architecture are similar to those realized from its use as a programming technique; the resulting protocol architecture minimizes the complexity and duplication of protocols and mechanism, thereby reducing the cost of implementation and verification. We also sketch a redesigned DoD Internet architecture that illustrates the potential benefits of this approach.

This work was sponsored in part by the Defense Advanced Research Projects Agency under contract N00039-84-C-0211, by Digital Equipment Corporation, by the National Science Foundation Grant DCR-83-52048 and by ATT Information Systems.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT □ UNCLASSIFIED/UNLIMITED □ SAME AS RPT □ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION	
22a NAME OF RESPONSIBLE INDIVIDUAL	22b TELEPHONE (Include Area Code)	22c OFFICE SYMBOL

83 APR edition may be used until exhausted
All other editions are obsolete.

Exploiting Recursion to Simplify RPC Communication Architectures

David R. Cheriton
Computer Science Department
Stanford University

Abstract

Current communication architectures suffer from a growing collection of protocols in the host operating systems, gateways and applications, resulting in increasing implementation and maintenance cost, unreliability and difficulties with interoperability. The *remote procedure call (RPC)* approach has been used in some distributed systems to contain the diversity of application layer protocols within the procedure call abstraction. However, the same technique cannot be applied to lower layer protocols without violating the strict notion of layers.

In this paper, we show how the RPC approach can be used for lower layer protocols so that the resulting "layer violations" generate a simple recursive structure. The benefits of exploiting recursion in a communication architecture are similar to those realized from its use as a programming technique; the resulting protocol architecture minimizes the complexity and duplication of protocols and mechanism, thereby reducing the cost of implementation and verification. We also sketch a redesigned DoD Internet architecture that illustrates the potential benefits of this approach. This work was sponsored in part by the Defense Advanced Research Projects Agency under contract N00039-84-C-0211, by Digital Equipment Corporation, by the National Science Foundation Grant DCR-83-52048 and by ATT Information Systems.

1 Introduction

Current communication architectures suffer from a growing collection of protocols in the host operating systems, gateways and applications. For example, an Internet host should implement, in addition to IP and TCP [15], the subtransport protocols ICMP [21], BOOTP [14], ARP [18], RARP [23] and now more recently IGMP [16]. The list continues to grow as new protocols are invented to handle more sophisticated management, query and exception handling functions. (The main data transfer portions of the architecture are surprisingly stable.)

The recent work of ISO on the *Open Systems Interconnection (OSI)* standard protocols is running into the same problem. The basic data transfer protocols represent only one portion of the architecture. The supporting management protocols represent a growing portion of the protocol suite.

This trend has several major disadvantages. First, the cost for the implementation and maintenance increases as new protocols are added, not to mention difficulties with interoperability. Second, the size of the implementations and the dynamics of interactions between protocols make reliability difficult to achieve and verification, such as might be required in a secure environment, impractical. Finally, the large number of protocols and size of code make providing hardware support to optimize protocol performance for the high-speed networks of the present and future almost impossible.

The *remote procedure call (RPC)* [2] approach has been used in some distributed systems to contain the diversity of application layer protocols within the procedure call abstraction and the suite of protocols used to implement RPCs. For example, file access, program execution, time service and remote database access can all be defined in terms of a set of procedures representing a module interface. The RPC system translates these procedure calls into (automatically generated) stub routines that use standard presentation, session and transport protocols for remotely invoking the services.

Lower layer protocols are also reasonably viewed as remote procedure calls. For example, RARP [23] is a specialized request-response protocol in the internetwork layer of the DoD Internet architecture that can be viewed as a remote procedure call that returns a host's IP host address, given its Ethernet address as a call parameter. Unfortunately, applying the RPC "solution" to lower layer protocols violates the conventional notion of layers, at least following conventional wisdom that communication architectures should be *strictly layered*[1]. However, using RPC at a layer below the RPC interface layer only results in the lower layer invoking the *RPC service interface* and not an arbitrary couplings to higher layers. The result is a *recursive architecture*, as illustrated in Figure 1. This structure is analogous to calling a procedure as part of the implementation of the procedure calling mechanism in a conventional programming language implementation, such as calling a procedure to allocate a stack frame as part of the procedure call mechanism itself. In this analogy, the whole RPC architecture is a procedure call mechanism and RPCs invoke the whole structure recursively as part of its overall implementation.

In this paper, we describe how recursion can be exploited in an

Also published in "Proceedings of SIGCOMM'88"

[1] The term *strictly layered* is used to refer to a layered architecture in which a layer may only invoke services of the layer directly below.

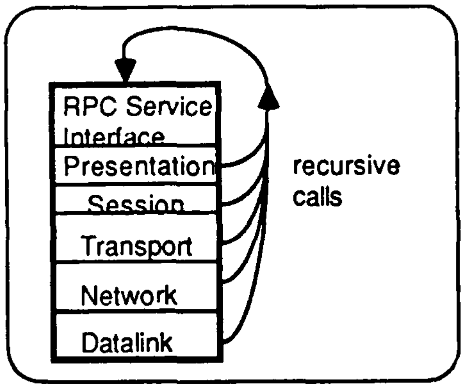

Figure 1: Recursive RPC Calls

RPC communication architecture to simplify the description, implementation and verification of the architecture. The unification and simplification of implementation makes hardware support for high performance protocol implementation significantly easier. In addition, we describe various techniques to ensure that recursive calls *terminate*. We also sketch a redesigned Internet architecture that illustrates the potential benefits of this approach, using VMTP [9, 8] as the transport protocol. The extended functionality of VMTP beyond conventional RPC, including multicast, datagrams, idempotency and priority is important, if not necessary, for a clean implementation of recursion.

The next section describes the use of recursion for simplifying the management portion of a protocol architecture. Section 3 describes the use of recursion to invoke query operations, such as arises in determining the network address of a server and self identity. Section 4 describes the use of recursion for the presentation level. In each of these sections, we identify the sources of potential unbounded recursion, and techniques to terminate the recursion. Section 5 describes how recursion facilitates the provision of hardware support to achieve high performance. Section 6 illustrates the use of these techniques by presenting a redesigned Internet protocol architecture that is considerabled simplified by the use of recursion. We close with general conclusions and discussion of open issues.

2 Management

Control, query and monitoring of protocol behavior are provided by a set of management operations implemented as part of the protocol module. Examples include operations to query the number of retransmissions, change buffering parameters and stop acceptance of incoming calls. We first consider how recursion simplifies access to management operations.

2.1 Access to Management Operations

Access to management operations below the presentation layer is conceptually a problem in a strictly layered architecture because the application cannot access the lower layers directly without violating the basic principles of layering and the management routines cannot be implemented at a higher layer within violating the integrity of protocol layer being managed. That is, these operations are an integral portion of the module implementing the protocol being controlled or monitored because they need to directly access and manipulate the protocol implementation data structures.

This problem motivated the provision in the original OSI architecture of a separate column of access for management which bypasses the normal layering, as suggested in Figure 2.

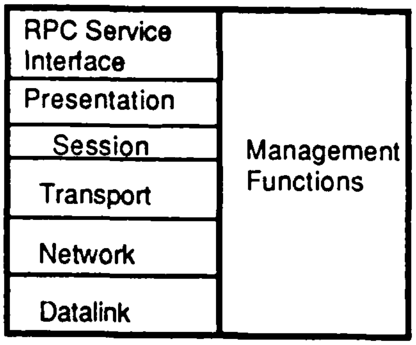

Figure 2: OSI Management Structure

However, exploiting recursion, the procedures of a protocol's management interface can be exported to the application level using the *export* service of the RPC service interface. That is, the module invokes the export facility of RPC service interface to export the management procedures as remotely invokeable procedures. Subsequently, these management procedures can be invoked by any modules with access to the RPC facility. Both the procedure export and the RPC invocation are illustrated in Figure 3. Note that the *export* service is a standard part of an RPC service interface, allowing each module to specify which of its procedures may be invoked "remotely". The export operation is a recursive call because a lower layer is calling the RPC service interface, which is at the application layer and implemented in terms of this lower layer.

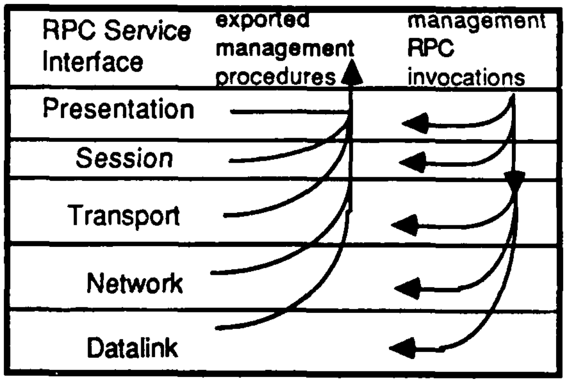

Figure 3: RPC Access to Management Procedures

This approach also makes these procedures available as RPCs to other protocol modules at the same or different layers of the architecture, whether they are running locally or remotely. In particular, RPC can be used by a management procedure to invoke operations in peer management modules on other hosts. For example, in VMTP, creating a (dynamically allocated) multicast group involves selecting a group identifier, checking that the identifier is not already in use and then adding a first member to the group. The checking phase requires communication with the other VMTP managers to ensure that the address is not already in use. Therefore, each VMTP management module exports a procedure that allows an RPC client to query whether a particular group has any members local to the exporting host. A VMTP management invokes this procedure as a multicast remote procedure call in all VMTP managers. As another use of this technique, the request to add this member must be communicated to its host machine, if the first member is remote from the requesting process.

Using this recursive approach, communication with the management operations and between management modules takes place using the standard RPC facility, requiring no special protocols. The recursive export calls do not repeatedly recurse because they simply add a record of the exported call to a local configuration data base or else communicate with a remote configuration database at a well-known address, as described in Section 3.

Exporting procedures to be called as RPCs can be use to handle other functions as well, an example being acknowledgements.

2.2 Acknowledgement Handling

Acknowledgement handling can be viewed as part of management, recognizing the control aspects of positive and negative acknowledgements. Positive and negative acknowledgements are required in an RPC transport protocol to handle several situation although, in the common case, the return packet acknowledges the call and a subsequent call or timeout acknowledges the return packets. As an example of the need for acknowledgements, consider a server sending a response to a client that has migrated to another host. The client's original host should send a notification to the server's transport module indicating that the response should be redirected to the new host. Conventionally, a special-purpose packet is used in the transport protocol to send a negative acknowledgement of this nature. However, exploiting recursion and the RPC export of management procedures, the notification can be accomplished as a RPC to the management module of the server, as illustrated in Figure 4. The number in the figure indicates the order of message transmission. First, the call request is sent followed by a response message, which prompts a migration notification call request to the manager of the server.

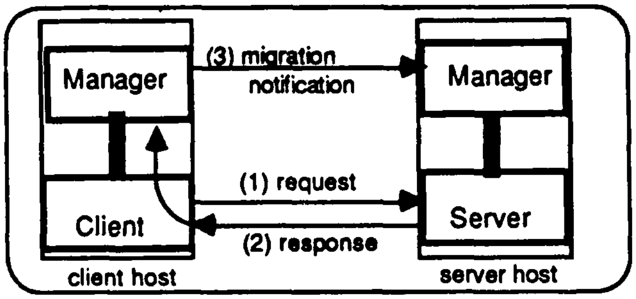

Figure 4: Notification/Acknowledgement as an RPC

Several optimizations on this basic approach should be supported by the RPC system. First, as suggested in Figure 4, the notification RPC should be sent as a datagram call because the invoking module does not require a response or normal confirmed reliable delivery. If the notification is lost, a subsequent event, such as the retransmission of the response, causes the call to be reinvoked, resulting in retransmission.

Second, these acknowledgement RPCs should be invoked with higher priority than normal RPCs so that an acknowledgement is not held up behind the transmission of user-level RPCs.

Finally, the acknowledgement call should be able to take advantage of the local "knowledge" of the host address for the manager, which is contained as a return address in the return or response packet that caused the notification to be sent. That is, the host address of the manager is known from the source address of the response and the fact that the manager is necessarily co-resident (on the same host) with the server. Without this support, it may be necessary to query the network to locate the manager.

To this end, VMTP, as an RPC transport protocol, supports datagram requests, priority, well-known multicast addresses and *co-resident* addressing to support all three optimizations. These extended RPC features are easy to implement and of wider utility, as described in Section 3.3.

2.3 Authentication Callback in Secure RPC

Management of secure RPCs involves authenticating a client and getting the encryption key to be used with the current and subsequent calls. These functions can be implemented using a *callback* to the client using a *challenge-response* protocol, as in Birrell's secure RPC [3]. That is, the server "challenges" the client to encrypt a random value; the client returns a response containing both an authenticator and the encrypted result. The random value protects against replays. The callback also eliminates the need to supply this extra information on every call. Callbacks are generally infrequent because the server caches the authentication information for a client between calls.

In Birrell's secure RPC protocol, the callback is implemented as special packets in the transport protocol. However, using the same techniques as described previously, we can instead export a management procedure `GetAuthentication` which is invoked by a recursive call from the server to the client's manager module, as shown in Figure 5. (The sequence of message transmissions is numbered in order 1 through 6.) The call to the authentication service is effectively another recursive call made by the client's manager as part of implementing the secure call. This authentication approach is used by VMTP [8] with the co-resident addressing mentioned previously to address the cuent's manager efficiently. The use of recursion again eliminates the need for special packets to handle the authentication callback

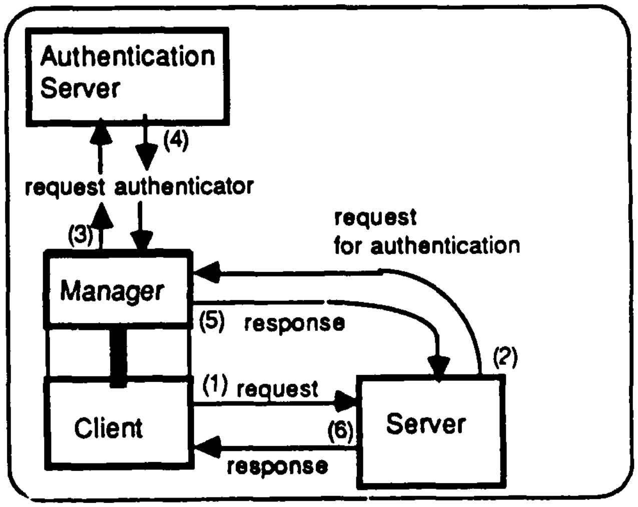

Figure 5: Recursive Call for Authentication

and takes further advantage of the RPC export of management procedures.

The callback call for authentication is actually performed as a non-secure call from the standpoint of the normal call encryption mechanism, thereby avoiding infinite recursion (to get the authenticator for the manager sending the callback). However, the callback is still secure because the call need not contain any parameters that have not already been sent in cleartext and the sensitive return parameters are encrypted by the authentication service or the respondent. Similarly, the call to the authentication server is made secure by using a public key for the authentication service to encrypt the call parameters and supplying a private key in the call that is used by the authentication service to return the response. *In essence, both these calls are made secure by special* case handling of the encryption of the call data and by restrictions on what is actually sent. These two mechanisms provide the base for the (recursive) implementation of the general secure call.

2.4 Exception Handling

An exception, whether a error condition or simply an unusual condition (as we have considered elsewhere [5]) often requires a sophisticated mechanism to properly handle the situation. It is attractive to make the full power of the RPC system available to handle exceptions in modules. However, this approach introduces another potential source of recursive structure when exceptions occur in the lower layers.

As one example, a module may use an RPC to remotely log when it receives a packet that contains a protocol error. Another example arises with a process incurring a page fault as part of a (remote) call invocation. On a diskless workstation, the page fault itself must be satisfied from across the network. Using a (recursive) RPC to read the page, the page can be retrieved the same as a conventional file read operation (without special protocols or mechanism). This recursion is particularly evident if the process is doing a remote file read at the point it incurs the page fault. The recursion terminates in this case because the process is now reading into a page frame which is of course never paged out.

In general, infinite recursion does not arise from the use of recursion for exception handling providing that we order all exception handlers and require that an exception handler only invoke exceptions that are strictly less than itself by this ordering. Typically, the handlers are ordered by increasing sophistication and the exceptions are ordered by decreasing severity. As a simple example, the transport module should never send a negative acknowledge to a negative acknowledge "call". To be more sophisticated, it should skip sending a negative acknowledge if the "severity" of the error code was less than that of the call to which it was responding. For example, one would not send a negative acknowledgement to indicate that the server had migrated in response to a negative acknowledgement such as described with Figure 4. The same reasoning applies to other management operations.

This explicit "architecting" of the recursive structure of the design makes the recursion safe and may well expose unintended recursion in design. The problem of infinite recursion with exception handlers arises independent of the use of the techniques described here. Every exception handler has to be concerned with incurring exceptions as part of its handling of the current exception. By exploiting recursion, we simply make this issue more evident.

The exporting of management operations as RPCs makes query operations available as RPCs that can be used for various binding operations, as described in the following section.

3 Binding Operations

Conventionally, specialized protocols are used for establishing bindings to remote servers as well as establishing local identity. For example, RARP [23] is used by a diskless workstation on the Ethernet to determine its IP address. However, these operations are *logically just remote procedure calls that return the* required information. The following subsections consider how to use RPCs for these binding operations without infinite recursion.

3.1 RPC Binding

The general problem for the client implementation of RPC is to bind an RPC stub to the right server and remote procedure, given a procedure p and object O. For example, p may be a file open operation on file O so the right server depends on the file name O. Alternatively, p may be a read from some open file O. As a special restricted case of this *object-oriented* binding, a procedure exported by a single server can be bound based only on the procedure name.

In the recursive approach, a binding is implemented as a remote procedure call that queries the binding from a directory server. That is, what is logically the session layer invokes the application RPC interface to access this directory server.

To avoid unbounded recursion, the directory server is addressed using a *well-known logical address*. Because the address is well-known, it is explicitly included in the session layer code so the code does not query (or recurse) to locate this server. This known value acts as the terminating condition for the binding in the same way as the known value $factorial(1) = 1$ terminates the recursion of the factorial function.

Administratively assigned multicast addresses, as provided in VMTP, are a good way to provide well-known logical addresses because they provide a level of indirection to the specific server, are easy to implement and allow for replication of the server. It is attractive to replicate the directory server for improved reliability and load sharing. In the following discussion, we assume the use of these multicast addresses.

Several optimizations on this basic approach arise. First, to avoid sending every query to all replicas of the directory server, the client can query the directory server group to locate a specific server, cache that specific server's identifier and use it until it is necessary to rebind because of server crash or overload. This optimization effectively introduces an extra level of recursion because the session layer query recursives to select a particular directory server when it does not have a valid directory server identifier in its cache.

Second, the scope of the multicast transmission to the directory server group can be limited in a large-scale system to a small subgroup, using (say) the time-to-live parameter in some protocols [10]. Thus, in the common case, only nearby directory servers receive the query. However, if the nearby directory servers have failed, the scope can be expanded to access more distant servers. In this fashion, the typical load on the network and directory servers for multicast queries is minimized.

Third, for objects such as files with a large name space and significant requirements for performance, reliability and security, the directory can be partitioned across the set of servers so that each server implements the directory information for its own objects. (This approach, as implemented in V [7, 11], allows the directory information to be made available with the same performance, reliability and security as the server because it is implemented as part of the server.) By caching information on which portions of the name space are implemented on the different servers, a client binds directly to the right server most of the time. On cache miss, the client (recursively) invokes either a multicast RPC query to the group of servers or a query to a directory server to determine the correct server. In either case, the cache miss results in an extra level of recursion.

More generally, the name space can be implemented as multiple levels of directory servers, as described by Lampson [20], rooted at a replicated *global directory server* and binding eventually to a local server that maintains directory information for its own objects, as described above for V. Each new level introduces a new level of recursion. For example, a query of the file name "%edu/stanford/dsg/bin/emacs" recursively queries on "%edu/stanford/dsg/bin", "%edu/stanford/dsg", "%edu/stanford", "%edu", and finally "%" with the last query satisfied by the hardwired binding of "%" to the well-known multicast address of the global directory server group. Caching reduces the expected amount of recursion to an insignificant level. For example, measurements of the V distributed system using the name cache [11] indicate name cache misses (resulting in name query operations) occur for less than 0.3 percent of the binding operations. As a consequence, the average cost of this recursive structure in the V naming system constitutes less than 2 percent of the average successful binding operation. In general, the use of recursion allows replication and partitioning the directory service across multiple servers with minimal mechanism. Name caching at each level results in good performance.

As a further optimization, some objects can be identified by a value that includes the server identifier as an embedded field so the server can easily be determined from the object identifier. An example is the tuple (server,local-obj-id) used to identify open files in V, called UIO objects [6]. With open files, this technique amortizes the cost of binding to the server over all operations on the open file, rather than doing a separate binding operation for all read and write operations. This further reduces the cost of recursion in the naming system. Embedding the server identifier in the object identifier also simplifies the allocation of object identifiers because every object manager can assign the second portion of the tuple independently, relying on the system-wide uniqueness of its server identifier to avoid collisions with other object identifiers. In V, identifiers of this nature are used for objects such as processes, address spaces, and other objects that are too transient to warrant assigning a character string name. (Extensions to the RPC stub compiler are required to take advantage of this technique.)

Finally, co-resident addressing, as supported in VMTP, can be used for operations that need to be bound to the specific server (in a group of servers identified by a well-known multicast address) that is co-resident with specified client. For example, the operation to get a client's authenticator is addressed to the client's manager which is co-resident with the client, as illustrated in Figure 5. This mechanism takes advantage of local knowledge of the host machine of the client in many situations so the procedure can be unicast directly to the correct manager module, avoiding the query operation to locate the manager.

The approach of assigning a well-known multicast group to a group of servers can be applied beyond its use for directory servers. For instance, there can be a well-known multicast address for each type of module. Members of the multicast group are the servers that implement this type. For each such object, the client queries the group (possibly in a type-specific manner) to determine the right manager and then addresses the call to that specific manager. The result is a *forest* of servers, with each server logically rooted by the well-known address for its type, as suggested in Figure 6. This approach further allows an object to migrate between managers of the same type providing that the object identifiers are unique across the type; the clients just rebind using the query mechanism when their cached notion of the specific manager for an object becomes incorrect because of migration.

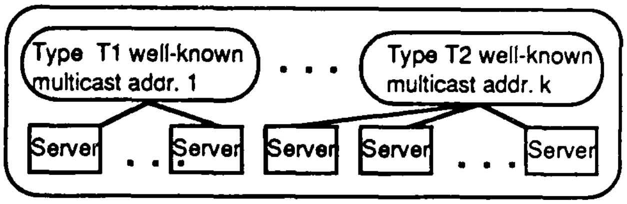
Type T1 well-known multicast addr. 1
. . .
Type T2 well-known multicast addr. k
Server
. .
Server
Server
Server
. . .
Server

Figure 6: Flat (Decentralized) Query Forest

This same technique can be applied to the binding and query problem at any protocol layer. For example, there can be a query operation at the transport level that determines the binding of a given transport address to host address, similarly for host address to datalink address and so on. The query operation can be exported by each management module the same as other management procedures, as discussed above. As a consequence, the functionality of specialized protocols such as RARP and ARP can be replaced by standard remote procedure calls.

Other attributes and parameters, such as maximum packet size, need to be set as part of communication with a server. However, a restricted version of RPC using default parameters is generally adequate for the simple query operations discussed above. In particular, both the call and the return parameters are relatively short so both generally fit into single packets.

Some environments may require a query to be done securely, with network intruders precluded from observing the contents of the query, modifying the query or responding to the query as an impostor. By (recursively) using a secure RPC facility for queries, the system can take advantage of security mechanisms in the RPC protocol, which should already be present and adequate in any environment that imposes such security requirements on the query operations. Without recursion, the security mechanism as well as the basic transport implementation would have to be duplicated as part of implementing the query mechanism. The normal secure RPC call can be used as soon as the client knows its (unique) client identifier. Establishing the client's identification is discussed in the next section.

3.2 Self Identification

A problem with using standard RPCs recursively to boot and initialize a client (machine) is that the client may not know its own "communication identity" initially. For example, consider booting a diskless workstation using the IP protocols. It needs to determine its Internet host address by querying the network. However, both TCP and UDP require that the workstation know its IP host address in order to use these transport protocols. In addition, it may need to determine its datalink layer addresses and other parameters of operation.

To allow use of the recursive approach in this situation, a communication entity uses *default* identifiers and addresses until it can determine or be assigned specific unique ones. In our example of the IP workstation, the workstation uses a default IP address initially. In general, each identifier space (application, process, host, gateway, etc.) must reserve a distinguished *default identifier* to be used in this situation. Thus, continuing our example, at the RPC level the workstation boot process acts as the *default client*, a well-known reserved transport-level client identifier. At the (inter)network level, the host uses a well-known reserved *default host address*. In addition, there are default values for the parameters associated with each protocol. In particular, at the RPC level, there is a *default call identifier*. The combination of the default client identifier and default call identifier defines the *default call* [2].

Several complications have to be handled to allow a client to use the standard RPC mechanism with default values. In particular, several nodes and processes may be using the default identifiers and parameters at the same time. Thus, two different call requests can come from two different network hosts with the same (default) client and transaction identifiers and be present on the network at the same time, making standard duplicate suppression unworkable. For *default calls* to work correctly, we require that each default call be handled as though *idempotent* and that the return parameters be *self-describing*, as defined below.

Handling a default call as idempotent means that the call processing is redone and a new response is generated every time a default call packet is received even though it may appear as a retransmission (given that every default call uses the same client and call identifier). The reprocessing ensures that the response matches the call parameters which are normally different between different default calls. Thus, each client call causes a response to be generate Because each default call can in fact be a different query, the response is not in fact idempotent but handling it in this way produces the desired behavior, namely a response specific to the call parameters. If the default call were not handled as idempotent, each subsequent default call would appear as a duplicate call and would generate a retransmission of the response to the previous call, defeating the use of the default call for name/address queries. Idempotent handling of default calls requires no special-case code in the servers if the transport protocol provides for idempotent responses, as in VMTP [9].

With multiple concurrent default calls in progress, there may be multiple return packets to default calls sent over the network in a short time range. Because one cannot guarantee precise routing of return packets, a default client may receive a return packet that is in fact a response to another node's call request. To handle this situation, we view return values to default calls to be essentially non-deterministic in that the return a client receives will be a valid return for some default call but not necessarily the one issued by the client. For example, a default client may ask about X but receive a response about Y. To handle this problem, default calls must have return parameters that are *self-describing* so the client can determine from the return parameters whether the response it receives matches the call request. If it does not match, the client discards the response and reissues its call after some timeout period. For example, the response to a query for the IP address for workstation with Ethernet address X returns the information "the IP address for X is H", rather than just "H". Thus, the node can repeat the query if it receives a response giving the Ethernet address for Y instead of that for X.

Ideally, there should be only one call that uses the default call so the client does not have to deal with multiple different return formats and self-identification schemes. However, to fully establish its identity as a communicating entity, a node must determine its identification and addressing at all levels, including the transport level, (inter)network level and possibly the datalink level. The order of determination that allows a single default call type depends on the protocol structure. For architectures such as TCP/IP in which the transport-level addressing is dependent on the (inter)network level addresses, the client should (first) use the default call to determine its (inter)network level address. It can then locally allocate transport identifiers and use its own unique identifiers. With a protocol like VMTP in which the transport identifiers are independent of the lower levels [3], the client (first) determines its transport identifier using the default call and then determines the bindings for the lower levels. Only the transport-level query needs to be self-describing and able to handle the incorrect responses that can be received to default calls. Once it has its own client identifier, a node can then proceed to generate unique transaction identifiers and therefore needs only one default call.

It is relatively easy to make this one simple query self-describing and allow it to be handled idempotently. The query does not change a server's state and the host usually has some unique identifier that it can send in the call to be returned in the response as an identification key for the caller. Examples of the latter include serial numbers and Ethernet addresses.

In the absence of a unique identifier to use on boot, a node must first allocate a unique number. One approach is to use random assignment from a large space (which minimizes the probability of collision), optionally checking with other nodes for collision. Interestingly, the check for collisions can be implemented as a default multicast query to all hosts. A response is expected only if there is a collision. Thus, there is no need to make responses to this query self-describing and so there is no additional recursion. Therefore, the multicast query call acts as the base (terminating) case for a recursive query structure for determining the host communication identification procedure.

A default call be performed securely if the configuration server that is to respond to these queries has a well-known public key, that is a *default* key. In this case, the default client uses this default key to encrypt its call parameters. It includes a private key in the call to be used by the server for encrypting the response. Only a valid server should be able to decrypt the call parameters and determine the private key so only a valid server is able to generate a response encrypted with the private key. The client decrypts each default call response it receives, discarding any that fail to decrypt correctly. The selection of the private key to use in this case is analogous, both in role and suggested mechanism, to the choice of a unique identifier for self-describing messages discussed earlier.

With the approach described above, an RPC call proceeds as

[2] This is a default transaction identifier defining a default message transaction in VMTP terminology [9, 8].

[3] Transport-level addressing that is independent of lower level addressing is important as support for process migration, multi-homed hosts, mobile hosts and accommodating different network-level protocols.

follows. On invocation of a call, the communication module first checks whether it knows the address for the server that is being addressed. If not, it (recursively) queries to locate the server. However, before performing either call, it first checks whether it is the default client caller. If so, and it is not already sending a query operation to determine its real address or identifier, it recursively invokes the query operation to determine its real identity before continuing with the original call. (The query operation is defined as a standard RPC call.) This recursive behavior is illustrated in Figure 7. In addition to the recursion shown in the figure, the client can recurse further to check for collisions when picking a unique identifier or private key[4].

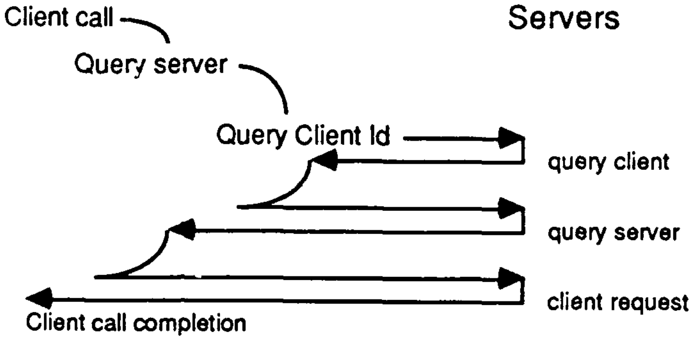

Figure 7: Recursive Calls as Part of a Client RPC

A response to a default call may have to be routed to multiple machines because several machines may be operating as the default client simultaneously. Therefore, the default identifiers are treated as multicast addresses. In particular, at the (inter)network level, the *default client host group* address is used as the default value, with this host group [10] corresponding to machines communicating as the default client. As an optimization, if a host that is operating as default client knows its lower-level identifiers or addresses, the server can record the low-level addresses associated with that default client call. Then, a response can be directed to the host originating the call using these low-level addresses.

A client switches from using default addresses to using its specific assigned addresses as it discovers these assignments. Servers should be prepared to rebind the addresses associated with a client as it begins to use these specific addresses. However, this rebinding is required to allow transparent migration of processes anyway so no new mechanism should be required. That is, a server must notice a new host address to associate with a client process after it has migrated if it already has a host address association cached for this process.

3.3 Extended RPC Functionality

Several of the techniques we have presented require extended functionality beyond that normally present in an RPC facility. This extended functionality is relatively easy to provide and of significant utility beyond its application here.

3.3.1 Multicast RPC to Well-Known Servers

We have assumed that there are well-known replicated servers implementing the directory and configuration services that are queried for server and client information. For this facility to work, all the layers must support the use of well-known logical addresses. By *logical address*, we mean an address that identifies a communication entity by its function or service, rather than by location. By *well-known*, we mean that these addresses are administratively assigned their particular logical meanings and can be safely "hardwired" into programs. Certain values can easily be reserved and administratively assigned in every identifier space used in the protocol architecture. The problem is mapping these values. Multicast addresses to provide logical addresses that are relatively easy to map and allow for replication of servers.

Well-known identifiers can be mapped using *well-known* mappings. For instance, a well-known transport identifier can have a fixed mapping to a well-known (inter)network identifier which has a fixed mapping to well-known network-specific identifiers. In a broadcast network such as the Ethernet. the network-specific identifier can be a multicast address that provides selective reception at the desired hosts. In a point-to-point or store-and-forward network, the network can provide *default routing* of packets addressed to the default address(es). For example, each switch may simply route each such packets out each outgoing link other than the one on which it was received. The use of caching, scope and embedded identifiers means that this relatively expensive routing need not occur frequently and need not extend over much of the total network if it is large. More sophisticated techniques have been developed [17] as well to handle internetwork multicast routing.

It is sufficient to have one well-known logical address that has a complete well-known mapping if that address is used for a directory server that provides access to all other addresses and mappings. However, a general multicast facility, as provided in VMTP, is useful for multi-destination delivery as part of replicated data update, for real-time state update and for various distributed algorithms, including scheduling, clock synchronization and atomic transactions. In fact, the other uses of multicast were the primary motivation for its development and use in VMTP and V.

3.3.2 Co-Resident Addressing

A second extension of RPC was to exploit *co-resident* addressing in conjunction with multicast. With *co-resident addressing*, a call is invoked at only those servers that share the same host (i.e. are co-resident) with an endpoint designated in the call. Co-resident addressing is implemented at the client end by looking up the host address corresponding to the specified endpoint in local data structures and transmitting the call to that host if the information is found. (Most of the circumstances in which co-resident addressing is used, this information is available locally.) If the host address is not found, the call is transmitted to the (inter)network multicast address corresponding to the transport multicast address. At the server end, any call specifying a co-resident entity that is not local to the server host is discarded.

Based on our experience in V and VMTP, this mechanism is easy to implement in an RPC system and results in efficient unicast addressing of managers without needing to first determine the specific identifier for each manager. It is also useful for a variety of situations in which it is appropriate to address one

[4] How to check for collisions of private keys without violating the security offered by the key is left as an exercise for the reader.

server out of a group that collectively provides the service for the whole cluster.

3.3.3 Idempotency

The self-identification problem required that the server specify in the response that the the call was *redoable on retransmission*, i.e. handled as though idempotent. From our experience with VMTP, this facility just requires a control flag in the response indicating that retransmissions should be handled in this fashion and the transport module checking this flag when it receives a retransmission. This flag also allows the response transmission code to discard the response once it is sent (because it will be regenerated by redoing the call if there is a retransmission). Overall, there is a modest amount of mechanism and insignificant overhead for this facility.

The idempotency facility is also useful for efficient file access support and for some real-time applications. For example, with file access, the transport layer of the file server need not incur the overhead of keeping a copy of the data blocks in case of retransmission. A retransmission simply accesses the data from the file server's buffer pool. In the case of real-time uses, the retransmitted response contains the latest data rather than what was sent in the previous response. For example, a call to get the current value of a sensor is better redone to get the new value if the original response is lost rather than retransmitting the old response.

3.3.4 Datagrams

The use of datagram calls is an important optimization in several of the situations considered. A datagram call is easily supported by the transport layer; a flag indicates that no response is expected and that no retransmission and timeout should be done. That is, it simply disables some existing mechanism rather than adding more mechanism.

The datagram call can be viewed as a conventional RPC that has no return parameters and is not guaranteed to occur. It is sufficient to have VMTP-like support for datagrams and a stub generator that allows certain remote procedures to be handled as datagram calls.

Datagrams are extensively used in real-time systems. Integrating datagram call with the RPC facility makes this important mechanism widely available.

3.3.5 Priority

Different priorities for calls are needed to cause negative acknowledgements to be handled responsively. Priority is also used for calls that implement routing, as described in Section 6. Implementation of priority requires a field in the transport layer header and priority-based transmission, reception and processing of calls according to priority. For example, a high priority call should be sent sooner than a lower priority call that is already queued for transmission.

Priority is also important in real-time applications in which response guarantees are important.

Overall, the extensions we advocate and assume in an RPC system to support the techniques described here are relatively easy to implement and provide functionality that is useful in a variety of other applications. Each of these facilities would be relatively straight forward to specify to a stub generator which then communicates these requirements to the transport layer.

4 Presentation

The *presentation* problem is to take an arbitrary procedure call and map it onto a standard (serial) network representation. We note that this mapping is normally defined in a recursive fashion to allow, for example, an array of arrays to be represented easily.

An example of the potential of recursion in the presentation protocol is the use of callback to implement a procedure parameter. Rather than defining how to transfer the procedure itself, the presentation level can require the recipient to call back the sender with a request to invoke the passed procedure. Then, the presentation layer need only specify how to represent the procedure call identifier, a much easier problem that describing the procedure itself. It also requires less conversion and transmission cost in general.

A second but similar example arises in the passing of large complex data structures. The server can recursively call back the client to get portions of this data structure as needed rather than passing it in its entirety at the time of procedure invocation. In both this case and the previous example, a well-known server (group) can be used to address these callbacks, with the server providing the invoking of these functions at the client end.

Finally, one can define a base presentation message format and then define all others (recursively) in terms of this base format or another so-defined format. For example, VMTP defines a basic presentation format to its messages as being 8 32-bit values followed by 0 or more octets in the so-called *data segment*. More complex data values are defined by their mapping onto this basic level, which in turn, maps onto the standard network representation. For example, a tree data structure would be mapped onto the 8 32-bit valies and the octets of the data segment for transmission. On reception, the receiver would map from this default presentation to its local representation of the tree data structure. The conversion between different machine representations of the basic message format would be handled by the lowest level of the presentation protocol implementation. The advantage of this approach is that the basic presentation format can be chosen to match the performance-critical case(s) and the implementation can then be optimized for this case, as described below.

5 Performance Benefits

A recursively structured RPC architecture defines the full-function RPC facility in terms of a more restricted version of itself. This suggests an "implementation" view of the layering of the architecture in which each higher layer implements an extended version of the same abstraction, as illustrated in Figure 8. Layer N provides full data representation including procedure parameters plus secure transport and authentication. It is implemented in terms of the more restricted versions of RPC provided by the lower layers. Conversely, layer i represents a level of RPC functionality as an extension of layer i-1. For example, a secure call is implemented in terms of an unsecure call. A non-idempotent call is implemented in terms of an idempotent call. Each layer implements some version of presentation, session and transport functionality. Layer 0 can be the idempotent, non-secure, non-duplicate suppressing, not fully reliable form of

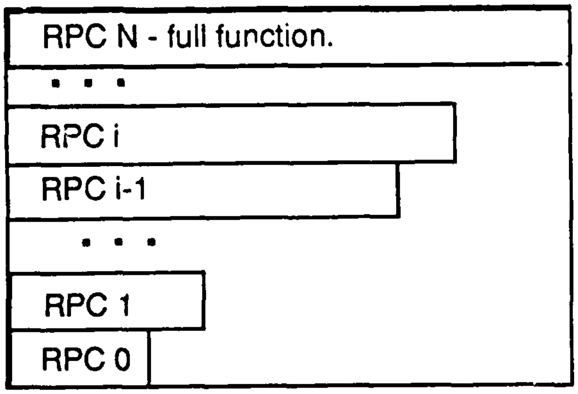

Figure 8: Recursively Implemented RPC Layers

call used to check for collisions with the choice of random boot identifier, as described as in Section 3.2. A significant difference between these layers and those of a normal architecture is that layer i in our model does not normally invoke layer i-1 as part of normal communication but only as a result of a cache miss or other unusual circumstances.

The VMTP and NAB [19] designs exploit recursive structuring to achieve high-performance communication using hardware support. The *Network Adaptor Board (NAB)* is a specialized board designed to provide hardware support for running VMTP over networks of 100 megabits per second or more. The design attempts to identify and support the most performance-critical functions of the protocol in hardware, focusing on packetizing, checksumming and encryption and their inverse functions.

The NAB supports in hardware the most performance-critical layer of RPC of the layers shown in Figure 8, dividing the layers into three major layers, as shown in Figure 9. This performance-critical layer is defined in our experience by the requirements of file read and write RPC operations [12]. Anticipating requirements for security, the common case is a secure call with a small number of short parameters, returning a similar number of parameters and a large parameter corresponding to the data to be read. A write operation is similar except the large data parameter, the data to be written, is sent as part of the call, not the return. In the common case, the server identification and parameters, client identification and parameters and encryption keys are known and cached.

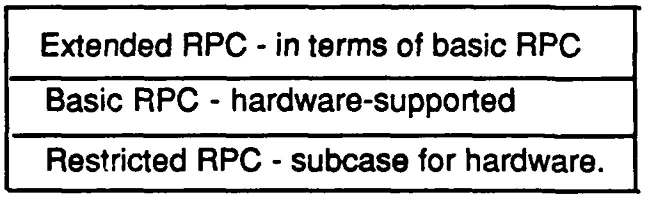
Extended RPC - in terms of basic RPC
Basic RPC - hardware-supported
Restricted RPC - subcase for hardware.

Figure 9: Extended, Basic and Restricted RPC

The NAB is designed to handle this common case call efficiently. In particular, it implements the restricted version of the presentation protocol used in this case. It also relies on getting the binding of server, client and encryption information (as required by the session level) from on-board caches. Finally, it packetizes, checksums and encrypts the data as required by the transport level, the reverse on reception. Other cases arise as a result of a cache miss or and because of a complex call.

In the case of a cache miss, the (software) cache miss handling uses a restricted version of the RPC functionality provided by the NAB to get the missing information. For example, a miss in the encryption key cache results in a non-secure call to the sender's manager. (Encryption support is not needed in this case as described in Section 2.3.) The hardware can readily support these simpler cases as a subcase of the common case call.

In the case of more sophisticated calls, the handling requires multiple RPC calls of the common case (or more restricted) calls or else extra processing at the sender and receivers. For example, a remote procedure call passing a procedure as a parameter may invoke multiple callbacks during its execution. As another example, a call format that is different from the common case format must be transformed into that format by the sender. In VMTP, for example, calls that match the common case presentation format of 8 32-bit words and a data segment of 0 or more octets, are transmitted in big-endian order. Any other call format must be transformed into this format before being transmitted by the NAB and transformed from this format when received. Thus, the representation of less common data objects must be defined in terms of the common data objects, not just a sequence of octets as is done conventionally.

Using this approach, the application user of RPC sees performance similar to that expected from a complete hardware realization but with a relatively low cost. That is, the hardware fully implements the common case RPC but only the common case. With large on-board caches, expected locality and repetitiveness in communication, the cache miss cases occur infrequently and, by definition, the other cases are also infrequent. The complexity of the less common cases are handled by software.

In contrast, using a conventional layered architecture, the hardware support generally implements fairly completely one or more of the low layer protocols, representing only a small portion of that needed for applications. As a consequence, hardware is wasted on supporting functions with no real performance benefits yet support is not provided for certain higher-level functions that are performance critical. Direct hardware support of common case communication appears essential to realize the performance potential offered by future high-speed networks.

6 Example: A Redesigned Internet Architecture

The potential impact of our recursive approach is further illustrated by sketching a redesigned version of the DoD Internet architecture using recursion to minimize the number of protocols and their complexity. A key part of this redesign is the use of VMTP as the transport protocol in place of TCP, as described below.

6.1 VMTP: The Transport Protocol

VMTP [8] is a request-response transport service tuned to RPC but augmented with support for multicast, datagrams, idempotency, priority and streaming. The inclusion of these facilities was motivated by application considerations such as real-time communication, efficient remote file access, and distributed parallel computation. However, these facilities are also useful, if not necessary, to support the recursive techniques, as described in Section 3.3.

The application of the recursive techniques of this paper to VMTP has lead to a protocol with only two types of packets (Request and Response) and the implementation of the mapping, management and exception operations on top of VMTP, using a standard RPC facility. (VMTP defines a standard representation protocol, procedure identifiers and a well-known entity group identifier (transport-level) multicast address for the VMTP management modules, sufficient for fully defining the binding and parameter formats for these calls.) In contrast to this simplicity, the early design of VMTP [9] used 8 different packet types and suffered as a consequence from complexity (and repetitiveness) of description and implementation. One pair of packet types corresponded to a "probe" query operation to determine the mapping of transport layer identifiers. This pair was eliminated by recursively invoking VMTP to perform the query, using well-known identifiers, as described in Section 3. This operation is also used to request and receive an authenticator as part of secure communication, as was illustrated in Figure 5.

A second pair of packets, the RequestAck and ResponseAck packets, were used for management and exception handling operations. These specialized packets were eliminated from the protocol by (recursively invoking) "notify" management operations as RPCs in the management module associated with the sender of the packet(s) being acknowledged.

These changes build on an original aspect of the protocol, namely exporting of the management module as a server so the operations could be invoked using VMTP. This management module implements operations for managing groups of entities (for multicast) and controlling servers. The extension of this module to handle the probe query operations and the notify operations was modest, and lead to a net reduction of mechanism in the protocol implementation.

As a result of using VMTP in place of TCP in our redesigned Internet architecture, the transport layer is a better base for application of recursive techniques, allowing us to further simplify the rest of the Internet architecture. It also appears easy to provide a high-speed implementation using hardware support such as supplied by the NAB.

6.2 Reducing the Number of Host Protocols

In the current Internet architecture, a full function host must implement a (growing) number of different specialized protocols in support of basic transport service. Examples include ICMP [21], ARP [18], RARP [23], BOOTP [14] and UDP [22] in addition to the prevalent transport protocol, TCP [15]. Each protocol requires its own procedures for transmitting, timing out, retransmitting and receiving packets.

The need for UDP is eliminated because VMTP provides a datagram facility. It also subsumes the other common use for UDP, namely the implementation of a request-response protocol, whether as a general-purpose protocol or as part of TFTP, NTP and other special-purpose protocols.

ARP and RARP are query protocols used to determine host addresses, the IP host address and the Ethernet address respectively. By making this information available through remote procedure calls, these protocols are replaced by recursive calls, with the caller using a default network or IP address for the call. This change replaces RARP with the use of the standard RPC mechanism, which is network-independent except for the specification of network address sizes (or type) and perhaps various default values as parameters in the calls.

The more recent BOOTP protocol is a query facility similar to ARP and RARP but operating on top of IP. Its functionality is similarly replaced by remote procedure calls. In fact, we argue that there need not be any special remote procedures for booting either; the required services can be supplied by a page-level file access interface, which is an obvious service to provide using the RPC facility.

ICMP is a subtransport management protocol for use with IP consisting of datagrams as well as request/response pairs. To eliminate ICMP, the IP module exports as RPCs the procedures corresponding to the handling of datagram notification calls: destination-unreachable, time-exceeded, parameter-problem, source-quench and redirect and the normal calls echo, timestamp and information. To address this service, the group of all IP management modules is addressed using a well-known transport-level multicast address and particular IP modules are designated using *co-resident addressing* (see Section 3.3.2) and a co-resident transport identifier derived from the IP address. The IP modules and other higher-level clients of ICMP are modified to invoke these remote procedures in place of sending ICMP packets.

6.3 Reducing/Unifying Network/Gateway Protocols

Gateways, routers and bridges can be viewed as servers, their services being communication interconnection between networks. Therefore, they can reasonably export an RPC procedural interface to their control and monitoring services. Thus, hosts and other clients can use the RPC facility to invoke these procedures to monitor, query and control the gateways.

Routing protocols are another example of specialized packet-level protocols that can be implemented using RPCs. These protocols consist of queries and notification calls some of which may be datagrams and multicast. Using RPCs for routing communication, the gateway can make greater use of the RPC facility it has to implement for monitoring and control. It also simplifies the specification of routing algorithms because communication is encapsulated as (remote) procedure calls. This approach appears applicable to both EGP and GGP.

Use of RPC and VMTP for routing protocols introduces further logical recursion, namely the routing of call packets used by the routing algorithm. That is, how does one route the call packet that is querying to find out a route. This case is handled by *default routing*, as described in Section 3.3.1. The default routing may simply correspond to broadcast or flooding the network. More selective routing follows once the information required for this selective routing has been acquired from queries using the default routing.

Implementing routing as RPCs means that each gateway must implement a relatively complete RPC facility. Fortunately, the memory and processing cost of a general-purpose facility of this nature is no longer a significant hardware cost, especially compared to 10 years ago when the Internet architecture was developed. Moreover, the same RPC support is then used for the monitoring and control procedures as well as for handling the remote procedures replacing ICMP. This use of an RPC facility in gateways can be exploited further, as described below.

6.4 RPC Gateways: Recursive Internetworking

Internetworking can be implemented using recursive RPC calls by extending the notion of transport-level gateways. Transport-level gateways were previously proposed by the author [4] as a solution to the performance, reliability and security problems with the internetworking of high-performance local networks[5]. In brief, to communicate with a remote endpoint (on another network), the client first creates a local *alias* endpoint in a local gateway representing this remote endpoint. The gateway makes communication directed to the alias appear as communication directly with the remote endpoint. For example, communication between two endpoints A and B on different networks takes place through gateway aliases, as depicted in Figure 10. As a consequence, the hosts need only understand the performance and administrative aspects of the local network. The gateway imposes access control between the local network and the internetwork. It can also (for example) tailor the retransmission rate to the delay and error rate of the internetwork link (by filtering out retransmissions). This structure also supports new techniques such as rate control [8, 13], which require hop-by-hop support for proper implementation. The interested reader is referred to the original article for more discussion [4]. This basic approach can be extended to provide RPC gateways with the intergateway calls viewed as recursive calls.

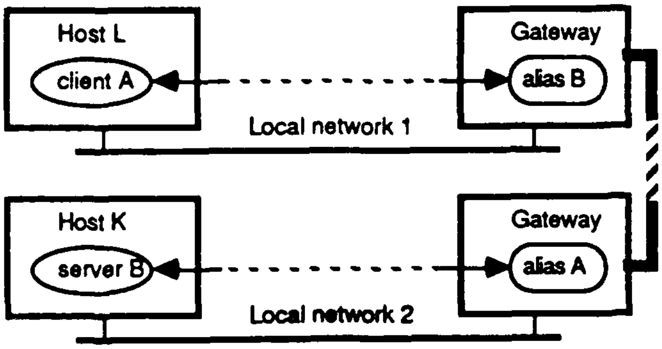

Figure 10: Transport-level Gateway Operation

Communication with a remote server is implemented as a sequence of recursive calls with each recursion corresponding to an additional hop between gateways. That is, a call to a server that is N networks away is implemented as a call to an alias for that server that is N-1 networks away. A call to a remote alias appears the same as a call to a remote process. Thus, a call from the local client to local alias recursively invokes a call from the local alias to the remote alias repeatedly for each gateway-to-gateway hop and from the "last" remote alias to the actual server. Each recursive call crosses a different communication domain, using potentially different naming, retransmission strategies, and protection for each recursive call. With each network representing a separate domain of *trust*, this approaches results in the same *relative* authentication and trust described by Birrell et al. [1]. However, the performance-critical mechanism at each gateway is simple because it only needs to handle communication within one network, further facilitating hardware support such as the NAB [19].

This technique can even be used to access and manage a gateway that is N-1 networks away. It is simply imported by recursively *importing* remote gateways into the local network using the alias mechanism. The recursive importing terminates when a gateway is imported that can communicate with the desired server in the configuration indicated in Figure 10. Importing a gateway allows it to be accessed and managed as though it were a local gateway[6].

The author is engaged in on-going work to develop this approach further to evaluate it as a credible alternative to current approaches to internetworking. A key issue is the reliability of internetworking with alias state in the gateways.

7 Conclusions

Recursion is a powerful technique for structuring RPC communication architectures. We have shown how various lower-level management, query and exception-handling services can be accessed remotely as remote procedure calls, using recursion to structure what would otherwise be a layering violation. The effect is to replace specialized protocols such as ARP, RARP and BOOTP in strictly layered architectures with procedural interfaces provided by the RPC system. These protocols are effectively part of the implementation of a full RPC facility, leading to a recursive structure. We showed how to apply these recursive techniques to the presentation, session, transport and network layer protocols, including routing protocols. We also showed how these techniques facilitate inexpensive hardware support. The application and benefits of this approach were illustrated by describing how the DoD Internet architecture might be redesigned and simplified using these techniques.

Compared to a conventional, strictly layered architecture, the basic service routines that implement the functionality of a specialized low-level protocol remain in the recursive architecture. The saving lies in the elimination of the packet handling code for each protocol and the special-purpose translation from procedure calls to communication packet formats. With an automatic stub generator, an increasingly common programming tool, even the code to generate and interpret transport layer messages is automatically generated from procedural interface specifications. The eliminated software, dealing with packet transmission, reception and timeouts, is significantly more complex for testing and verification than the procedure interfaces resulting from our approach. Thus, these changes reduce the overall size and complexity of what is characteristically the "networking software".

Using our recursive approach, the conventional architecture layers for application, presentation, session, transport, network, datalink and physical remain intact. Arbitrary calling into the higher levels is forbidden and most calls continue to be from one layer to the layer directly below. Lower layers are only allowed to use the RPC service itself (the highest level service interface) and any procedures that are exported through the RPC facility by other modules. Thus, strict layering is violated but the loss of modularity is minimal. In particular, lower layers only incorporate knowledge of the interfaces of the exported remote procedures they use and their ability to invoke these procedures. They otherwise remain ignorant of the protocols, service interfaces and implementations of the higher-levels allowing these intermediate-

[5] In this approach, multiple physical local networks are connected by bridges rather than gateways to form a single *logical local network* if there are no performance or administrative boundaries between them.

[6] This technique is used in an implementation of transport-level gateways used by the Port PC networking system developed by Waterloo Microsystems.

level protocols and implementations to be replaced transparently, a claimed advantage of a strictly layered architecture. The major problem is guaranteeing the absence of unbounded recursion.

Recursion does not impose a significant performance penalty because recursion is not used on the critical path in the common case. Typically, recursion only arises on a cache miss or with less common operations. Moreover, the recursive approach results in all layers using the same transport protocol and protocol implementation. This unification has significant performance benefits when hardware support is provided (and needed) for high-performance transport service, as the NAB [19] provides for VMTP.

Our recursive approach also structures the full RPC communication facility as successive extensions of the same basic abstraction. This structure appears to facilitate verification of the security and reliability properties of a communication architecture compared to a layered architecture in which each layer is a totally different abstraction. In particular, recursion leads to more succinct description of the architecture, reducing the amount of specification (and programming). The resulting short description facilitates proof of correctness as does the recursive structure, which is often amenable to an inductive proof. Further work is required to evaluate the merits of recursion in simplifying the verification task.

Well-known values are used to ensure that recursive calls terminate. For example, well-known multicast addresses are used to address servers (or discover the address of servers) whose specific server address is not already known. The well-known addresses provide a terminating condition for recursive query operations called to map to server identifiers and addresses. Also, well-known default values are used by query operations whose purpose is to (recursively) determine specific values for communication parameters, including the identification of the caller. Although these default values limit the functionality and reliability characteristics of RPCs using these parameter values, we have argued that the functionality is adequate for bootstrapping.

In conclusion, the growing sophistication of communication environments and services requires that there be a consolidation in protocols. The remote procedure call approach offers such an opportunity. Handcrafted, special packet formats and packet generating and handling can be replaced by automatic techniques analogous to the replacement of handcoded assembly subroutine calls by compiler-generated calling sequences. While these benefits are largely recognized at the application level, similar benefits accrue at the transport level and lower if one accepts the recursive techniques advocated here. In any case, some new approach is required to stem the tide of new specialized protocols within each protocol architecture. Our investigation of the recursive approach presented here finds it very promising.

8 Acknowledgements

The formulation of these ideas has benefited from discussions with Steve Deering, Carey Williamson, Ross Finlayson, Hemant Kanakia, Marc Abrams, Michael Malcolm, Jon Crowcroft and Lorenzo Aguilar.

References

[1] A.Birrell, B.Lampson, R.Needham, and M. Schroeder. A global authentication service without global trust. In *Proc. Symposium on Security and Privacy*, pages 223–230, IEEE, April 1986. IEEE Computer Society order number 716.

[2] A. Birrell and B. Nelson. Implementing remote procedure calls. *ACM Trans. on Computer Systems*, 2(1), February 1984.

[3] A.D. Birrell. Secure communication using remote procedure calls. *ACM. Trans. on Computer Systems*, 3(1), February 1985.

[4] D.R. Cheriton. Local networking and internetworking in the V-System. In *8th Data Communication Symposium*, IEEE/ACM, 1983.

[5] D.R. Cheriton. Making exceptions simplify the rule (and justify their handling). In *World Congress*, IFIP, 1986. Dublin, Ireland.

[6] D.R. Cheriton. UIO: A uniform I/O interface for distributed systems. *ACM Trans. on Computer Systems*, 5(1):12–46, February 1987.

[7] D.R. Cheriton. The V Distributed System. *Comm. ACM*, 31(3):314–333, March 1988.

[8] D.R. Cheriton. *Versatile Message Transaction Protocol (VMTP)*. RFC 1045, SRI Network Information Center, February 1988.

[9] D.R. Cheriton. VMTP: A transport protocol for next generation communication systems. In *SIGCOMM '86*, ACM SIGCOMM, August 1986.

[10] D.R. Cheriton and S.E. Deering. Host groups: a multicast extension for datagram internetworks. In *9th Data Communication Symposium*, IEEE Computer Society and ACM SIGCOMM, September 1985.

[11] D.R. Cheriton and T.P. Mann. Decentralizing a global naming service for efficient fault-tolerant access. *ACM Trans. on Computer Systems*, 1988. To appear; An earlier version is available as technical report STAN-CS-86-1098 Computer Science Department, Stanford University, April, 1986 and CSL-TR-86-298.

[12] D.R. Cheriton and C. Williamson. Network measurement of the VMTP request-response protocol in the V distributed system. In *SIGMETRICS '87*, ACM, 1987. Banff, Canada.

[13] D.D. Clark, M. Lambert, and L. Zhang. *NETBLT: A Bulk Data Transfer Protocol*. RFC 969, SRI Network Information Center, 1985.

[14] B. Croft and J. Gilmore. *Bootstrap Protocol*. RFC 951, SRI Network Information Center, March 1985.

[15] DARPA. *DOD Standard Transmission Control Protocol*. IEN 129, SRI Network Information Center, January 1980.

[16] S.E. Deering. *Host Extensions for IP Multicasting*. RFC 1054, SRI Network Information Center, May 1988.

[17] S.E. Deering. Multicast routing in internetworks and extended LANs. In *SIGCOMM '88*, ACM SIGCOMM, August 1988.

[18] D.Plummer. *An Ethernet Address Resolution Protocol*. RFC 826, SRI Network Information Center, November 1982.

[19] H. Kanakia and D.R. Cheriton. The VMP network adapter board NAB: High-performance network communication for multiprocessors. In *SIGCOMM '88*, ACM SIGCOMM, August 1988.

[20] B. Lampson. Designing a global name service. In *Proceedings of the Fifth Annual ACM Symposium on Principles of Distributed Computing*, pages 1–10, ACM, August 11-13 1986. Calgary, Canada.

[21] J. Postel. *Internet Control Message Protocol*. RFC 792, SRI Network Information Center, September 1981.

[22] J. Postel. *User Datagram Protocol*. RFC 768, SRI Network Information Center, September 1980.

[23] R.Finlayson, T. Mann, J.Mogul, and M.Theimer. *Reverse Address Resolution Protocol*. RFC 903, SRI Network Information Center, June 1984.

END

DATE

FILMED

11-88

DTIC