CONNECTING RPC-BASED DISTRIBUTED SYSTEMS USING WIDE-AREA NETWORKS*

Robbert van Renesse
Andrew S. Tanenbaum
Hans van Staveren

Department of Mathematics and Computer Science
Vrije Universiteit
Amsterdam, The Netherlands

Jane Hall

Computer Science Division
Hatfield Polytechnic
Hatfield, England

ABSTRACT

Remote Procedure Call (RPC) is a widely used communication mechanism in local network based distributed operating systems. It is simple, fast, and straightforward to implement. However, when two or more distant distributed systems are connected, problems arise concerning the protocols, locating services, and other issues. To solve these problems, gateways are introduced. In this paper we discuss various ways in which these gateways can be organized and show how their application in the Amoeba Distributed Operating System has solved the problems cited above.

1. INTRODUCTION

As networks of high-performance personal workstations become more widespread, interest in distributed operating systems to make the whole system look like a single time-sharing system is increasing. When the same distributed operating system is running on two widely separated local-area networks, it is natural to think about merging them into a single transparent distributed system. However, because local-area and wide-area networks have very different properties, a number of problems arise. These problems and some proposed solutions are the subject of this paper.

A brief outline of this paper follows. First, distributed operating systems are discussed, in particular those properties which make them hard to extend to wide-area systems. Next gateways are discussed. Several types of gateways are distinguished using the ISO OSI reference model [1]. Then attempts at transparent wide-area extension are described for the Cambridge Distributed Computing System, the V-System, and the

*This research was supported in part by the Netherlands Foundation for the Advancement of Pure Research (Z.W.O.) under grant 125-30-10.

Amoeba Distributed Operating System. Finally the solutions are compared.

This work was undertaken as part of the COST-11 ter MANDIS project partially sponsored by the European Community. In this project, two Amoeba Distributed Operating Systems in Holland (Vrije Universiteit, CWI), two in the U.K. (Harwell, Hatfield Polytechnic), and one in Norway (University of Tromsø) are being connected into a single, transparent distributed system as a research project, with the aim of investigating the tools and services required when interconnecting autonomous management regimes.

2. DISTRIBUTED OPERATING SYSTEMS

When computer networks first appeared, the operating systems used on the computers were just ordinary operating systems extended with networking primitives. Using these primitives it was possible for a process on one computer to set up a connection or *virtual circuit* to a process on a remote computer. This communication was not transparent because the syntax and semantics of intramachine communication were different from intermachine communication. Such a system is called a *network operating system*.

The next evolutionary step in this direction was to try to hide the machine boundaries, so that to the user, the collection of machines would look and act like a single multi-user time-sharing system instead of a collection of autonomous machines. This led to the concept of a *distributed operating system* [2] in which all the machines ran the same operating system kernel and handled resource management automatically. For example, in a distributed system, when a process or file is created, it is up to the operating system, not the user, to decide where to place it.

These differences in approach also led to differences in protocols. Network operating systems tend to do infrequent bulk file transfers, which can best be handled by connection-oriented sliding window protocols. On distributed operating systems, processes tend to have frequent short interactions with other processes, leading to connectionless *remote procedure call* [3] as the most widely used communication model.

Wide-area networks, like network operating systems, generally use connection-oriented protocols such as the ISO OSI protocols. When two RPC-oriented distributed systems need to communicate over a wide-area system, problems arise due to the different communication styles.

Another important property of distributed systems is how they locate processes and services. If a client, C, calls a server, S, the system must have some way of locating S. One approach is to have a central name server that maps S onto the machine number where S is located. The other approach is to broadcast a message asking all machines on the network if they know where S is. Neither of these approaches is suitable when extending a distributed system to multiple remote sites. Something else is needed.

3. GATEWAYS

Whenever two networks are connected, a gateway machine is needed between them [4]. The gateway has to deal with problems caused by

- different name spaces
- different packet sizes

- different protocols
- support of broadcast

In addition, gateways can play a role in flow control, congestion control, service location, and protection. In this section we will see how gateways deal with these problems, and on which level of the ISO OSI reference model they are taken care of. We shall see that it is not possible to put all gateway functionality below the transport layer, as proposed by OSI. An advantage of putting the gateway as low as possible in the layer hierarchy is that higher-level software does not have to differentiate between different networks (since there is only one logical network). However, we will show that it is not necessary for higher-level software to be aware of different networks even if gateways are at the highest layer.

On the lowest level in the OSI hierarchy is the *connector*, a simple, theoretical, gateway that physically links up the underlying cables of the network. Somewhat higher in the physical layer is the *repeater*, which amplifies the signal before transfering it to ‘‘the other side.’’ *Selective repeaters* filter out messages that are not intended for the other side. They are in the data-link layer. Both connectors and repeaters have the property that they are completely software transparent, even where timing is concerned.

On the data-link level we find *relays*, which receive complete packets and transfer them to the intended networks. Relays are also software transparent when timing is not critical. High in the data-link layer we find relays that connect physically different networks, and make them look like a single physical network. The packet size on this network is the minimum of the packet sizes of all the cooperating networks, since the relay does not do fragmentation. Addresses of the new logical network are mapped to physical network and site address using routing tables. Usually, however, this is done on higher levels.

Network-layer gateways support different networks, and they fragment packets if they are too large for the destination network. Network-layer gateways can also help to avoid congestion in the network by re-routing packets.

Transport-level gateways understand transport protocols, and can ‘‘adjust’’ them for different networks, for example, by filtering out retransmissions that arrive too fast for the destination network. Moreover, they can do address translation: the local address is mapped to a remote address. Local addressing can therefore be independent of network-wide addressing. These gateways are implemented by having the gateway ‘‘stand in’’ as the remote process, that is, the local process thinks that it is talking to another local process. In reality, the other local process is a half-gateway that transfers messages to and from another half-gateway on the remote network.

Session-level gateways work at the session layer and above. Here they have full control over participating networks, simplifying management and enhancing flexibility. Communication over these gateways is efficient, since the participating networks can be optimized for their local characteristics (effectively they do message reassembly). This is especially true if the networks differ considerably in bandwidth, and there is not much to be gained in forwarding packets before all have arrived. Flow control problems are taken care of on either side of the gateway.

4. IMPLEMENTATIONS

Many local-area networks are currently in use. However, only a few of them have been connected transparently over a wide-area network. The advantages of connecting these local systems are obvious—connecting them transparently allows applications to work unchanged across local network boundaries. In this section we will have a look at some of these systems, and how they have been implemented.

4.1. The Cambridge Distributed Computing System

The Cambridge Distributed Computing System [5] is an experiment of the University of Cambridge to provide a computing system consisting of processors connected by a fast communication network, the *Cambridge Ring*. Some of the system's processors perform dedicated services, such as a name service or a file service, whereas others form a multiple-purpose *processor bank*.

4.1.1. Single Site

The unit of communication between the processors is the *basic block*, consisting of a source address, a destination address, and a chunk of data. Several end-to-end protocols have been built on top of these blocks. For example, the Single Shot Protocol (SSP) is a simple communication interface to exchange request and reply messages. Services are named by character strings. The name server maintains the location of all services.

When a process wants to make use of some (local) service, it sends a NAME-LOOKUP-REQUEST to the name server to find the location of the service. The name server itself is situated at a well-known address. It sends a reply containing the address of the server back to the original process, using the source address in the basic block that contained the request. Now the process can send requests to the service using SSP-REQUEST messages.

4.1.2. Multiple Sites Connected By A Wide-Area Network

To allow communication with services provided on different rings, rings can be connected by *bridges*. A bridge is a special processor on the ring having its own ring address, or two ring addresses if it connects two rings directly. The bridge serves two functions. First, it helps in locating remote services, and second, it transfers basic blocks between rings. This last property makes the bridge a data-link level gateway. All this is transparent to the client process. These protocols have been applied successfully in the UNIVERSE Project [6, 7, 8] which combined several local-area networks using satellite and X.25 wide-area networks.

Remote contact is established as follows. As usual, the process sends a NAME-LOOKUP-REQUEST to the name server. The name of the ring where the server resides is specified in the request. For now we assume that the name server also knows the destination address on that ring. It then sends an ADDRESS-INSERTION-REQUEST to a bridge, which allocates some data structures and acknowledges the request immediately. Then the name server sends a special reply to the client process containing the address of the bridge, and the global address of the server.

The client software then sends an SSP-REQUEST to the bridge, thinking that the bridge is the service. The SSP-REQUEST is automatically converted to a BRIDGE-SSP by inserting the global address of the service. The bridge forwards the BRIDGE-SSP to other bridges using static routing tables, until the destination ring is reached. Each bridge remembers the source address so that a reply can be routed back. The last bridge transforms the BRIDGE-SSP into an ordinary SSP-REQUEST, and sends it to the final destination. Replies are returned using the backward path set up by the bridges, and delivered to the original client process.

The forward path and backward path may now be used by the client process and the server process transparently. When the client thinks it is sending a basic block to the server, it is really sending it to a bridge which forwards it to the destination network. The bridge at the destination network then sends the basic block to the server process which will think that it received the message from the client. The forward and backward paths do not provide any flow control or error correction; this must be taken care of by the end-to-end protocols.

Now suppose that the name server does not know the destination address of the server process. Now the name server sends a REMOTE-NAME-LOOKUP-REQUEST to the name server on the destination ring. The destination address of the name server is fixed on every ring. The local name server sends a request to the remote name server in the same way as a client process would send a request to a remote service. The REMOTE-NAME-LOOKUP-REPLY from the remote name server contains the destination address of the wanted server. The local name server caches the remote address for possible future reuse and sends the client the address of the bridge on the local network in the NAME-LOOKUP-REPLY. Again the client and server are completely unaware of the bridges between their networks.

4.2. The V-System

The V-System [9] is a distributed operating system running on a collection of processors connected by an Ethernet. The processors are divided into two types: workstations and server machines. The workstations are like processors in the Cambridge Processor Bank, except that workstations have owners that have high-priority access to their machines. The server machines provide services like file access and printing.

4.2.1. Single Site

Interprocess communication is through request-reply exchanges: a client process sends a request to a server process, and then awaits the reply subsequently sent by the server. The protocol used here is developed specially for this purpose for optimal performance, and because no suitable standard interprocess communication protocol was available [10].

In V, services are named by character strings. To locate a service, the client *broadcasts* the name of the service over the network (broadcast is a special case of V *multicast*). The service replies with a *process identifier*—a location dependent number that identifies a process—and from thereon contact is established. This scheme has been extended for any kind of object by using directories on each processor, forming a *global naming directory*.

4.2.2. Multiple Sites Connected By A Wide-Area Network

Internetwork communication in V *should be* implemented as follows. We say ‘‘should be,’’ since the current implementation is somewhat simplified. When a process wants to access a remote service, it broadcasts the service name as usual over the local-area network. The gateways on the network forward this message to all the the remote networks, where it is then re-broadcast. The remote service replies as usual, thinking that the gateway on its network is an ordinary client process. This gateway sends the reply back to the local network where the real client resides. The local gateway sends the reply on to the client, which, again, thinks the gateway is the server. The local gateway starts a *local alias process*, a pseudo-process that represents the server. In the same way, the remote gateway starts a *remote alias process* to represent the client. All messages sent between the client and server are forwarded by these alias processes over the wide-area network.

One problem arises since the process identifiers are local to a network. The process identifier of a remote service has no meaning on the local network. To solve this, the process identifiers have to be translated to local process identifiers when passed to the local network. The file server therefore has to be aware of this, by returning the process identifier of the remote alias process, violating transparency.

(The current implementation of the gateway does not support internet broadcasting, making the service locating protocol unusable for locating remote services. Instead, when a process wants to access a remote service, it first has to request the gateway to create a local alias process for the remote service.)

V gateways know about the higher-level protocol, and use this knowledge to optimize communication over wide-area networks (without affecting local networking). For example, if two gateways are connected by a reliable virtual circuit, the gateways can filter out end-to-end acknowledgements, and generate them locally instead. This does not violate end-to-end reliability in V, since all requests are acknowledged by replies in the end. Another optimization done by V gateways is the combination of packets over virtual circuit connections. Furthermore, retransmissions that arrive at a rate too high for the virtual circuit are discarded.

The knowledge of transport protocols makes V gateways at least transport-level gateways. However, the gateways have also higher-level properties. Before forwarding messages, V gateways inspect them to ensure that local security policies are not violated. This way the local network becomes a *security domain*. V gateways thus allow for access control, authentication, and accounting. By implementing these checks in the gateway, instead of at every site, local performance is not affected, nor are the local security policies.

4.3. The Amoeba Distributed Operating System

The Amoeba Distributed Operating System [11, 12, 13] is a research project being carried out at the Vrije Universiteit and the Centrum voor Wiskunde en Informatica, both in Amsterdam. Amoeba is also based on the client-server model. Server processes provide services like file and directory service. Amoeba runs on a collection of Motorola 68010 and 68020 processors connected by 10 Mbit networks.

Processes in Amoeba are addressed by *ports*. Ports are location-independent 48-bit numbers. A process can choose any port it wants to. By taking a random unique 48-bit number, servers can have a private address that they can use on any machine. It is even possible to use the same port for more than one process. This way a service can increase its availability by replicating its server processes. The communication protocol selects one of them.

4.3.1. Single Site

A client process invokes a service by sending a request message to the server process. When the server has executed the request it returns a reply message to the client. These request-reply exchanges are called *transactions* [14]. They are used as a basis for implementing remote procedure calls.

When a client process starts a transaction, the server has to be located first. This is done by broadcasting a message containing the port of the server process over the local-area network. The machine running the server sends a reply back containing its network address. This information is cached by the client machine, so if it needs the same service in the near future, it can try the same network address without having to broadcast again. If this address turns out to be wrong (servers and their ports can migrate), it can resort to broadcast to locate a server again.

Once the network address of the server is known, a simple protocol optimized for the local network ensures reliable transmission of the request and reply messages. Moreover, this protocol ensures at-most-once delivery of requests, avoiding problems that occur when requests get executed twice due to retransmissions [15].

4.3.2. Multiple Sites Connected By A Wide-Area Network

Extending the transaction implementation to wide-area networks meets with difficulties. First, the scheme of locating ports by broadcast does not work with multiple Amoeba systems connected by a wide-area network. Wide-area networks usually do not support broadcast. Simulating it by sending the messages separately to each site is expensive, even when minimum spanning trees [16] are used. Furthermore, a broadcast causes overhead on each site of the wide-area network. Second, the protocol that is efficient for local-area networks is inefficient for wide-area networks. Also, since the only access to the wide-area networks is through high-level interfaces, the low-level Amoeba messages are transported using a high-level protocol.

A solution to the first problem is presented through *publishing*. Servers can publish their port and wide-area network address in the *domains* where they want to be known. A domain is a set of Amoeba sites that are logically related. As soon as a port has *appeared* in an Amoeba site, processes in this site can use the server.

To enable this, a process is created for each port that appears. This process will *stand in* for the remote server by using the port as its own Amoeba address, and is therefore called the *server agent*. If a client process tries to locate the remote server, it will find the server agent instead, and a request message for the server is sent to the server agent. The server agent forwards the request across the wide-area network using the published wide-area network address.

When the request arrives at the remote Amoeba site, a process is created there to send the request message to the server. This process, called the *client agent*, starts a local transaction with the server. The server thinks it received a request from another client process.

The reply it returns is sent by the client agent to the server agent. The server agent then forwards the reply to the real client, completing the transaction.

All this is transparent to both the client and server processes. Moreover, it is transparent to the kernels that run the Amoeba transaction protocol. This implies that the transaction protocol is unaware of the existence of the wide-area network, and that no unnecessary overhead exists for local communication. In the same way, the wide-area network protocol knows nothing about transactions, but just forwards complete messages. The client and server agents together form the gateway, which is called a *transformer* [17]. A transformer is a session-level gateway, since it uses the transport protocol interface. Flow and congestion control are provided by the wide-area network, so the transformer does not have to do it. As it is at a higher level than the network layer it ‘‘knows’’ more about these interactions and can provide valuable information about distributed processing across network boundaries.

Publishing is implemented by a server running at each site, together forming the Service for Wide-Area Networks, or SWAN. Each SWAN server listens to a well-known port, and can therefore be easily contacted from anywhere in the world. A port is published by doing a transaction stating port and wide-area network address with each SWAN server in the required domain. Each SWAN server will then automatically start a server agent.

5. COMPARISON

Having discussed solutions to wide-area networking for several different systems, we will compare their properties in three categories: naming transparency, protocol transparency, and gateway functionality. Each of these categories will be discussed in the following sections.

5.1. Naming Transparency

This section discusses whether client processes have to know the location of a service, and whether services have to know the location of the client when passing names in replies. These properties are highly dependent on the local naming strategy.

In the Cambridge system, local services are named by ASCII strings. Names are mapped to local network addresses using a central named server. This scheme is extended to wide-area networking by prefixing the service name with the name of the ring. For example, suppose there are two networks called A and B, each having a printer server called *printer*. A client process on network A can access the local printer using the name *printer*, and the printer server on B using the name *B*printer*. The name server knows which bridge to use to reach B, and the wide-area network address of B. Finding services given their name is not a problem anymore; however, there is no naming transparency. Also, if a process on B wants to pass the name of the printer server to a process on A, it will have to prepend B to the name.

V uses a decentralized naming approach relying on broadcast. In the V-system, the printer server on B should have a different name from the printer server on A, since they do not provide exactly the same service (they use different printers). This scheme is transparent. However, V uses a two-level naming scheme. A process can not send a message to a remote process using the remote process identifier.

Amoeba has an implicit decentralized naming scheme using ports. The port space is not local to a network, and therefore ports can be passed freely from network to network. As in V, the printer servers would have different names. However, since Amoeba gateways do not support broadcast, a port that is passed from A to B has to be published in B first. Since this is different from local operation, Amoeba does not provide full naming transparency either.

5.2. Protocol Transparency

In this section we will see to what extent the local protocols are affected by supporting wide-area communication. It is important that the performance of local communication is not degraded, since local communication will represent the bulk of all communication. Changing the local protocol software may not be possible if it is a commercial package without source code. Even if the source code is available, it has to be avoided since it requires all sites on the network to change their system.

The Cambridge system uses the local transport protocols over wide-area networks. This was made possible by using a fixed internetwork packet format (basic blocks). A disadvantage of this approach is that these protocols may not be suitable for wide-area communication. For example, the timeouts that are used in SSP are adjusted for remote services.

V nodes are unaware of gateways, and the local protocol is unaffected. When a remote process is referenced, a local alias process takes care of transfering messages to and from the remote network. As in the Cambridge system, the local protocol was also used for internetwork communication.

Amoeba also provides full protocol transparency, but unlike V, Amoeba does not use the same protocol for wide-area communication. Instead it uses whatever protocol is available on the wide-area network. This means the Amoeba machines can use protocols optimized for the local cast on local networks and the gateways can use other protocols over the wide-area network without the clients and servers knowing about it.

5.3. Gateway Functionality

The different solutions to wide-area communication require different gateway functions. The functionalities of the gateway determine its complexity. The gateway in the systems we discussed have two important tasks. One is to help in locating servers, and the other is to provide transparent communication between processes separated by a wide-area network.

Cambridge bridges help in locating services by forwarding OPEN-REQUESTs. This is a simple operation, using static routing tables. The bridges remember the paths they formed, so that they can use them for forwarding basic blocks. The transport protocols are unknown. Since the bridges just forward the basic blocks, it is possible that for the wide-area network to become flooded or congested, so flow and congestion control may be needed.

V gateways, on the other hand, are fully aware of the transport protocol. For service location, they have to set up minimum spanning trees to forward broadcasts. Gateways optimize the protocol to avoid flooding the wide-area links. Furthermore, gateways check messages to prevent them from violating security constraints. All this makes a V-gateway a complex device.

In Amoeba, the naming and protocol problems are solved separately. The SWAN service takes care that ports are distributed where they are needed. The transformer achieves protocol transparency by transfering complete messages, based on routing information provided by the SWAN. The transformer uses the transport interfaces of both Amoeba and the wide-area networks, and therefore does not know the transport protocols. Nevertheless the protocols are designed especially for the type of network they are running on. There is not much to be gained in forwarding packets instead of messages if local and wide-area networks differ considerable in bandwidth. Both the SWAN service and the transformer are simple devices.

6. CONCLUSIONS

In this paper we discussed how distributed operating systems, designed for local networks, can be connected into a wide-area distributed system. Two problems were identified. One is how to locate a remote service over a wide-area network. The other is providing transparent communication. We described how these problems were solved in the Cambridge Distributed Computing System, the V-System, and the Amoeba Distributed Operating System.

All these systems use special gateways to transfer messages between the local networks. The data-link level gateway used in the Cambridge system supports several transport protocols, but the protocols do not adapt well to wide-area networking. The V transport-level gateway does protocol optimization, since it knows the transport protocol. This makes it better suited for wide-area networking, but it is also a complicated gateway.

The Amoeba gateway uses the transport-level interface of both the local network and the wide-area network, and is therefore a session-level gateway. The gateway is not concerned with how to provide efficient communication, but leaves that problem to the local and wide-area network software. By *standing in* for remote processes, it provides transparent communication without affecting local networking. Services make themselves ‘‘known’’ through *publishing*, that is, they relate their existence to all local networks in which they want to provide their service.

It is argued that the Amoeba gateway provides the same gateway functionality as other gateways, and that the implementation is more efficient. Furthermore, in the normal case when the bandwidth of the wide-area network is considerably lower than that of the local networks, the performance is at least as good as that of other gateways.

7. ACKNOWLEDGEMENTS

We would like to thank Gojko Babić, David Holden, Jack Jansen, Bram Janssen, Kari Lang, Martin Turnbull, and Wolfgang Zimmer for valuable discussions. We also thank Jennifer Steiner for the careful reading of this paper.

8. REFERENCES

References

1. H. Zimmermann, ‘‘OSI Reference Model—The ISO Model of Architecture for Open Systems Interconnection,’’ *IEEE Trans. Comm.*, vol. COM-28, pp. 425-432, April 1980.
2. A. S. Tanenbaum and R. van Renesse, ‘‘Distributed Operating Systems,’’ *ACM Computing Surveys*, vol. 17, no. 4, pp. 419-470, December 1985.
3. A. D. Birrell and B. J. Nelson, ‘‘Implementing Remote Procedure Calls,’’ *ACM Trans. Comp. Syst.*, vol. 2, no. 1, pp. 39-59, February 1984.
4. C. A. Sunshine, ‘‘Interconnection of Computer Networks,’’ *Computer Networks*, vol. 1, no. 3, pp. 175-195, January 1977.
5. R. M. Needham and A. J. Herbert, *The Cambridge Distributed Computing System*, Addison-Wesley, Reading, MA, 1982.
6. C. J. Adams, G. C. Adams, A. G. Waters, I. Leslie, and P. Kirk, ‘‘Protocol Architecture of the UNIVERSE Project,’’ in *Pathways to the Information Society (Proc. of the 6th Int. Conf. on Computer Communication)*, ed. M.B. Williams, pp. 379-383, North-Holland, Amsterdam, September 7-10, 1982.
7. I. M. Leslie, R. M. Needham, J. W. Burren, and G. C. Adams, ‘‘The Architecture of the Universe Network,’’ *Computer Communication Review*, vol. 14, no. 2, pp. 2-9, 1984.
8. S. R. Wilbur and P. T. Kirstein, ‘‘The Universe Catenet: its Protocols and Issues,’’ *IEE Proc., Part E*, vol. 132, no. 4, pp. 189-195, July 1985.
9. D. R. Cheriton and W. Zwaenepoel, ‘‘The Distributed V Kernel and its Performance for Diskless Workstations,’’ *Proc. of the 9th ACM Symp. on Operating Systems Principles*, pp. 128-140, Bretton Woods, NH, October 1983.
10. D. R. Cheriton, ‘‘Local Networking and Internetworking in the V-System,’’ *Proc. of the Eighth Data Communications Symp.*, October 1983.
11. S. J. Mullender and A. S. Tanenbaum, ‘‘The Design of a Capability-Based Distributed Operating System,’’ Report CS-R8418, Centre for Mathematics and Computer Science, Amsterdam, October 1984.
12. S. J. Mullender and A. S. Tanenbaum, ‘‘Protection and Resource Control in Distributed Operating Systems,’’ Report IR-79, Dept. of Mathematics and Computer Science, Vrije Universiteit, Amsterdam, August 1982.
13. A. S. Tanenbaum, S. J. Mullender, and R. van Renesse, ‘‘Using Sparse Capabilities in a Distributed Operating System,’’ Report IR-100, COST-11 ter/DSM MANDIS (VU): 2, Dept. of Mathematics and Computer Science, Vrije Universiteit, Amsterdam, March 1985.
14. S. J. Mullender and R. van Renesse, ‘‘A Secure High-Speed Transaction Protocol,’’ *Proc. of the Cambridge EUUG Conf.*, Cambridge, UK, September 1984.
15. A. Z. Spector, ‘‘Performing Remote Operations Efficiently on a Local Computer Network,’’ *Comm. ACM*, vol. 25, no. 4, pp. 246-260, April 1982.
16. Y. K. Dalal, ‘‘Broadcast Protocols in Packet Switched Computer Networks,’’ Ph.D. dissertation, Computer Science Dept., Stanford University, Stanford, CA, April 1977.

17. R. van Renesse and J. M. van Staveren, ‘‘Wide-Area Communication under Amoeba,’’ Report IR-117, COST-11 ter/DSM MANDIS (VU): 5, Dept. of Mathematics and Computer Science, Vrije Universiteit, Amsterdam, December 1986.