THE LIBERTY NET: AN ARCHITECTURAL OVERVIEW

I. NASSI
ONTEL Corporation
250 Crossways Park Drive
Woodbury, NY 11797
(516) 364-2121

Abstract

This paper presents an architectural overview of an operational local area network of workstations and servers. The relationship among capabilities, encryption, and authentication is explored in detail. The concepts of ports, referential transparency of network communication, and kernels as processes are discussed.

Introduction

The Liberty Net project is an attempt to investigate several architectural components of a general purpose local area network of personal computers and shared central resources. Rather than tailoring the system for a specific environment, such as an office, we assumed an environment that was heterogeneous both in function and in physical configuration. Thus, it is possible in the Liberty Net to connect workstations that are economically well matched to the function to be performed without being constrained by preconceived hardware configurations. On the Liberty Net, we expect to see secretarial workstations, programmer workstations, electronic mail workstations, teletex stations, and timesharing systems, as well as shared resources such as file servers, high quality document preparation facilities, a central post office, gateways to other networks, and network monitoring and control nodes.

Because of the requirement for a heterogeneous system and the desire to support a network configuration that could rapidly change, we chose to make it possible for each node to be reasonably autonomous. That is, each node is responsible for some limited set of functions, and if necessary it can then negotiate with other nodes for required services. Thus, there is a requirement for establishing a basis for that negotiation that is secure and can be implemented on a variety of computers. Further, we felt that "referential transparency", the property of shielding a communicating process from the knowledge of the location of the process it was communicating with, would provide additional flexibility in partitioning multiple process programs on physical systems.

Interaction among processes in the Liberty Net is accomplished by means of message passing. Associated with each conversation is a client, which initiates the transaction, and a server which responds. Clients can simultaneously be servers for other transactions, and vice-versa. Clients and servers might be implemented as single processes, but can often consist of a number of cooperating processes as well.

The method of interprocess communication is via ports, and is described in the next section. The extension of the port model across the network is accomplished by means of a "net server" process, which is described next. The means for naming services and authenticating clients and servers is then defined, and finally the current implementation is discussed.

Ports

Following the ACCENT system,[1] the principal method of interprocess communication (IPC) is via "ports". A port, as implemented on a physical node in the Liberty Net, is a protected kernel object that is known to a process by means of a local name, i.e. a capability. Processes send and receive messages on ports. Messages themselves can contain ports as components. Possession of a capability for a port is an implicit right to send messages to that port, and only processes that have that right can exercise it.

When a port P is sent in a message, the process receiving the message automatically receives the right to send messages to P. At most one process can receive messages from P and, unlike ACCENT, that right cannot be transferred.

To insure that a process only sends and receives from the proper Ports, the actual port data is kept by the kernel in a per process table. A process then represents the port as an integer index into its port table. Thus, when the kernel receives a request from a process to perform an operation on a port, it must translate the specified local name into a port. That translation, although quite simple, is a function of the process making the request. When a port is passed in a message, translation is also required. For example when process A sends port 17 to process B, B receives a port which it might refer to as port 42.

CH1796-2/82/0000/0099$00.75©1982 IEEE

In order to efficiently perform local name translations, all messages must have a well defined structure known to the kernel. In our system all ports can be discovered by a self describing message format, without having to parse each message individually. Generally, all communications with the kernel, other than sending and receiving messages, are handled by message passing; the kernel can thus be viewed as a distinguished process. These messages to the kernel, analogous to system calls on other systems, are used to manage process and memory resources for that node. Because communication with the kernel is controlled by message passing, the structure permits processes on one node to control processes on another node, and to act as a surrogate kernel in a straightforward way.

A description of the operations on ports appears in the Appendix.

Net Server

Ports as described in the preceding section only work on a single node. Ports can be extended transparently across a network (i.e. the property of referential transparency) by the following method. The net server is a process running on each node. Its purpose is to convert interprocess communication on the node into interprocess communication over the network, while maintaining the port abstraction. In order to accomplish this, a change in the representation for ports is needed to manage the port name space in the absence of a centralized control. In contrast to the operating system kernel, which generally uses hardware protection modes, the net server relies on encryption as a means to provide a secure port abstraction.

We make the assumption that each physical node has an address, and that at each physical node there are a set of sockets[2, 3]. The network representation of a port is a triple, (N, S, K), where:

N	is a node number
S	is a socket number
K	is an encryption key

The net server maintains a table of ports and, like the kernel, performs a mapping of local port names to their network representation, and the inverse function. The encryption key associated with a port is the key used to encrypt all messages sent to the port, and to decrypt all messages received on the port. Since all network messages on the Liberty Net are sent to ports, all Liberty Net network messages are encrypted.

There are two kinds of operations performed by the net server. First, messages can be received on a network socket; these will be decrypted and forwarded to the corresponding local port. Second, messages will be received on local ports; these must be encrypted, and sent on the corresponding network socket.

Ports can be sent in messages. When the net server receives a message containing ports on a local port, send rights to those ports must be transmitted over the network. If after checking its internal tables the net server determines that this is the first time it has seen this port, a network address for the port is allocated, an encryption key is allocated, and an entry is made for the port in the internal table. This entry contains the encryption key, the node and socket addresses, and the name of the local port. The message is modified so that the network representation of the port that is being sent in the message is substituted for the name of local port. Note that the newly allocated key is being sent embedded within a message, and so is itself encrypted under another key, namely the key of the port to which it is being sent.

Just as ports can be sent in messages, ports can be received in messages as well. When a message containing ports is received on a network socket, the table is again consulted. In this situation, send rights to the ports contained in the message are being received. If the network address already appears in the table there is already a local port; the name of the local port is substituted for the network port and the message forwarded to its destination. If the network address is not found in the table, it is entered along with the encryption key that was passed. A local port is created, and entered in the table as the surrogate of the network port. The internal table contained in the net server is then modified by adding this port to the set of local ports on which it expects to receive messages.

The effect of these translations, and the embedding of encryption keys in messages results in a multi-level key distribution scheme, in contrast to the approach of[4].

Ports can be deallocated implicitly by process completion or by a call to an explicit port deallocation service. In either case, this raises the possibility of a server becoming stuck in an infinite wait (e.g. on a port that no longer exists). Requiring that all waits have time out periods is not a good solution to this problem because of the difficulty of establishing an appropriate timeout period in an internet environment. To solve this problem we have introduced a NO_SENDERS exception that is raised when a process is waiting for a message on a port, and there are no senders to that port. The exception is raised in the context of the receiving process after all send rights have expired, and all messages to that port have been received. This exception is used in a net server to effect clean up of its port correspondence tables. When the net server detects this condition for an outgoing surrogate port, the net server deallocates its local port, and using the corresponding network address, forwards the exception over the net to the net server on the other end, at which point the remote net server deallocates its corresponding local port. It is easy to imagine the NO_SENDERS exception rippling back through the network causing ports to be cleaned up when appropriate.

In summary, processes send messages to the ports of other processes. If the processes are on different nodes, the net servers on both the local and remote ports cooperate to effect the transfer in a way that is transparent, except for timing delays, to both the sender and to the receiver.

Authentication

Central to any resource control scheme is the need to accurately identify the source of service requests. In a local area network such as this one, there are two choices: each server implements its own authentication scheme, or a single authentication scheme is mandated for the entire network. The latter approach was chosen for the Liberty Net; however, each server is free to refine the basic scheme according to its need.

In this scheme there is a single, trusted, distinguished server known as the authentication server[4]. This has the disadvantage of introducing a potential single point of failure for the entire system. However, as will be seen, its structure is very simple and could be implemented on a dedicated, highly reliable microprocessor. Alternatively, it could reside on a node like the central file server, which if unreliable has a major impact on the successful operation of the system anyway.

This authentication server differs from the one described in[4] in two ways. First, it does not supply encryption keys or nonce identifiers. Encryption keys are provided by the network port model. Second, it combines the functions of authentication and name services[5] into a single service. It serves to register a service, deregister a service, service a "request for service", and provide for network log in. Each of these will be described in turn.

Assume for now that the authentication server maintains a single port called a "private port", for each independent cooperating set of processes that require an identity (for example, all the processes belonging to JOE). We will call this set of processes a "citizen" of the network, and the method by which these ports are initially established will be defined as part of the network log in function.

When a server wishes to advertise its services on the network, it sends a message to the authentication server on the server's private port, indicating that it wants to register some service by name, and passes the authentication server a port to be used on which to receive requests. The server trusts that the authentication server will never share send rights on that port.

The authentication server registers that name for the server by maintaining a correspondence table of services (represented by their names) and the ports that were passed with the service registration messages. Note that the server's identity is implicit; the act of receiving the request on a port associated with the server is enough to guarantee that either it was the server who sent the request in the first place, or else it was someone who was appointed by the server.

Deregistration of a service simply removes the corresponding table entry.

A client who wishes to make use of a service sends a service request message to the authentication server; the message contains a port (the "reply port") on which the client expects to receive a reply message containing a port on which to request services. The authentication server can identify the client implicitly by receiving a message on the private port associated with the client. The authentication server looks up the named service. If the service isn't found, the client is so informed on the reply port on which the client expects a response. If the named service is found, the authentication server sends a message to the port associated with the service, containing the identity of the client, the client's request, and the reply port. The server receives the message from the authentication server, and can assume the request was from the authentication server because the server trusts the authentication server not to grant any other citizen send access to this port. If the server wishes to grant service to the client, it allocates and sends a port, the "service port", to the client's reply port. The service port is the one on which the client can talk to the server. If the server decides not to grant service, it also can inform the client on the client's reply port. Any messages subsequently received on the service port can be assumed by the server to have either come from the client, or a process the client appointed as a surrogate, i.e. by passing along its send rights.

The authentication server maintains a "public port" for clients to log in on. A public port is like any other port on the network except that all citizens have send rights to the public port by knowing its node number, socket number and encryption key. Indeed, it really need not have an encryption key, but that would mean it was handled in a manner that was different than any other port in the system. The authentication server also maintains one "master" encryption key analogous to a password for each citizen of the network. Recommended practice dictates that the establishment of the master keys not be done over the network, but be established by an "out of band" channel. For example, the system administrator may physically unlock the authentication server and manually key in the required information directly.

Revisiting the net server for a moment, recall that the net server is a process running on each node whose purpose is to convert messages on the node into messages over the network. On initialization, a net server sends a message on behalf of a citizen to the public port of the authentication server, requesting that a private port be established. A reply is passed to port with the message. The authentication server responds to the reply port with a message containing the private port for that citizen. This initial transaction is completely normal except for one additional item: the reply message from the authentication server is encrypted, not only under the normal encryption key provided with the reply port, but with the master key known only to the authentication server and to the citizen. Only when the master key is known, can the private port the authentication server has allocated for the citizen be discovered. The use of the master encryption key prevents a citizen from tricking the authentication server and every other server in the network into believing it is a different citizen.

After initialization, the net server's port correspondence table has a network port in it, namely the private port to the authentication server. The encryption key used is the one expected by the authentication server, and so it could be thought of as an interchange key. There is also a local port, which the net server has receive rights to, that is logically interposed between the authentication server and the network port. This local port can be made available to other processes acting on behalf of a citizen on the node for use in communicating with the authentication server provided of course that those processes belong to the same citizen. Note that the master key itself need never be sent over the network. This scheme avoids all hardwired ports except for the single public port on the authentication server.

Liberty Net Enhancements and Limitations

Nothing in this design precludes multiple net servers from operating on a single node. This seems to be the desirable way to implement gateways.

With very minor modification, this scheme will support nodes with multiple identities (such as time sharing systems). All that is needed is to initialize the net server with one private port for each identity. Each identity receives a local port with which it communicates with the authentication server.

The possibility of a distributed authentication server requires further exploration, but appears to be feasible.

The Liberty Net design cascades gracefully. By caching ports, servers can get into an extended dialogue with clients if they so desire, without having to communicate with the authentication server again. Caching ports will reduce the workload on the authentication server dramatically. For security reasons it may be desirable to keep transactions short so that keys are changed frequently.

This proposal does not address the possibility of replayed messages, messages received out of sequence, or corrupted messages (above the normal communications checksum). Given the model that permits send rights to be transferred this freely, keeping track of sequence numbers presents some difficulties. Sequence numbers and additional checksums could be part of higher level protocols if this turns out to be necessary. Replayed messages can be handled by nonce identifiers, as in (4).

Implementation Status

A prototype of the Liberty Net has been built in DEC's Corporate Research Group. The system consists of five LSI 11/23s running UNIX*, and interconnected by 10MB Ethernet. As of this writing, plans are being made to add VAX-11/750s** to the Liberty Net. In addition to workstations, there is a post office, a central file server, and the authentication server.

As expected, without IPC modifications to UNIX, and with processors of limited speed, performance is poor.

Security goals for the system were modest. Rather than building a secure system, we wanted to concentrate on key distribution strategies. As a result, we decided to use a computationally inexpensive encryption algorithm. It would clearly be preferable to use an encryption chip, as is now currently available from a number of commercial suppliers, given the amount of encryption being performed. Alternatively, if security is not an issue for a specific application, no encryption need be applied except for log in.

* UNIX is a trademark of Bell Telephone Laboratories.

** VAX is a trademark of Digital Equipment Corporation.

Acknowledgements

This design and implementation was the collaborative effort of a number of people: Ed Balkovich, Ken King, Dave Leblang, Pete Lee, Fred Maryanski, Dan McCue, Craig Schaffert and Dave Velten. It benefited from discussions with Rick Peebles, Roger Needham, Maurice Wilkes, George Robertson and Rick Rashid.

Our port model was influenced heavily by the Accent system under development as part of the Spice Program at Carnegie-Mellon University.

References

(1) R.F. Rashid and G.G. Robertson, "Accent : A Communication Oriented Network Operating System Kernel", Proceedings of the Eighth Symposium on Operating Systems Principles, December 1981, pp. 64-75.

(2) DOD Standard Transmission Control Protocol, January 1980, Defense Advanced Research Projects Agency, 1400 Wilson Boulevard, Arlington, Virginia, 22209.

(3) XEROX Corporation, "Internet Transport Protocols", XSIS 028112, December 1981, Xerox Corporation, Stamford, Connecticut 06904

(4) R.M. Needham and M.D. Schroeder, "Using Encryption for Authentication in Large Networks of Computers", CACM 21, 12 (December 1978), pp. 993-999.

(5) A.D. Birrell, R. Levin, R.M. Needham, and M.D. Schroeder, "Grapevine : An Exercise in Distributed Computing", Proceedings of the Eighth Symposium on Operating Systems Principles, December 1981.