Extending a Capability Based System into a Network Environment

Robert D. Sansom, Daniel P. Julin and Richard F. Rashid

24 April 1986

Abstract

The Mach operating system supports secure local communication within one node of a distributed system by providing protected communication capabilities called Ports. The local port-based communication abstraction can be extended over a network by Network Server tasks. The network servers effectively act as local representatives for remote tasks by implementing an abstraction of Network Ports. To extend the security of the port-based communication abstraction into the network environment, the network servers must protect both the messages sent over the network to network ports and the access rights to network ports. This paper describes in detail the protocols used by the network servers to support protection.

Technical Report CMU-CS-86-115

Table of Contents

1. Introduction

Systems such as *DEMOS* [3], *Hydra* [1] and *Accent* [8] pioneered the use of capabilities to provide access to communication channels. Communication channels protected in this way - DEMOS links, Hydra mailboxes and Accent ports - could in turn be used to represent protected access to resources or service objects managed by user-level processes. Several commercial offerings, for instance the MBOS operating system developed by ELXSI Corp., have also followed this strategy. The flexible protection mechanisms offered by the coupling of communication channels with capabilities have allowed these systems to implement a variety of security policies within a single computer node.

While techniques for implementing capabilities in a single computer system have long been established [9], little has been done to extend capability systems - especially those based on message communication - beyond a single uniprocessor or tightly coupled multiprocessor into a network environment. With the exception of Accent, network operating systems research has tended to emphasise connectivity and communication speed over security issues. The V Kernel [4], for example, supports extremely fast network transmission times for messages but provides neither security for the transmission of data by a process nor protection from malicious data transmitted by other processes.

As the size of networking environments have grown from tens to thousands of nodes and as network operating systems have begun to make the transition from research tools to production computing environments, the need to extend local node security mechanisms, such as capabilities, into the network environment has become apparent. At Carnegie Mellon this need for increased security has led us to extend the original Accent model of transparent network communication to include the secure exchange of communication capabilities and data. This work has been incorporated into a new multiprocessor operating system which is the logical successor to Accent: the *Mach* kernel [2]. Mach is 4.3 BSD UNIX[1] binary compatible and has been adopted within the CMU Computer Science Department as the standard multiprocessor and uniprocessor operating system on VAX mainframes and workstations. As of March 1986, Mach was operational on the four processor VAX[2] 11/784, as well as members of the MicroVAX[2] family.

The Mach kernel's capability system is defined in such a way that it can be extended transparently into the network environment by user-level *Network Server* tasks. Such an extension offers many advantages: it supports higher-level protection mechanisms for the distributed system; it maintains the semantics of the local capability system in the distributed system environment; and it naturally provides secure communication channels between tasks on different nodes of the system.

---

[1]Unix is a trademark of AT&T Bell Laboratories. [2]VAX and MicroVAX are trademarks of Digital Equipment Corporation.

In this paper we will show how the protocols used by Mach's network servers maintain the same security guarantees over the network as the Mach kernel provides within one node of the system. The network server protocols protect both communication between tasks on different nodes and access rights to ports. They thus provide a secure network communication abstraction which can be used as the basis for higher-level security protocols such as authentication and authorisation of users and servers. These higher level security services are discussed in a companion paper [10].

2. Local Capability Scheme

The Mach operating system kernel supports a secure local Inter-Process Communication (IPC) abstraction based on communication capabilities called *Ports*. Ports are kernel objects on which messages can be queued or dequeued. *Tasks* are the basic unit of resource grouping in Mach and can only access a port if they hold a local capability for that port. Since tasks execute in protected virtual memory address spaces[3], they can only affect other, unrelated tasks by using the communication facilities. The Mach kernel protects access to ports and allows tasks to hold one or more of the following access rights to a port:

- *receive* - only one task at a time may hold receive access to a port. A task with receive access to a port is allowed to dequeue messages that have been queued on the port.

- *ownership* - only one task at a time may hold ownership access to a port. A task with ownership rights to a port will acquire the receive rights to the port should the receiver[4] die. Should the owner die, then the receiver acquires the ownership rights. By default, the ownership and receive rights are held by the same task. However, splitting the ownership and receive rights provides some degree of fault tolerance.

- *send* - many tasks may hold send access to a port. A task with send access to a port is permitted to queue messages on the port provided that the port's queue is not full. When both receive and ownership rights to a port are deallocated, the Mach kernel informs senders of the *death* of the port.

A task can only gain access to a port by receiving the access rights in a message sent to it by the kernel or by another task that already holds rights to the port. Send rights to a port can be transferred by a sender as well as the port's receiver or owner. Furthermore, both receive and ownership rights can move between tasks.

The Mach kernel provides three basic security guarantees for ports and messages:

1. Only the task holding receive rights to a port can obtain a message sent to that port.

2. Only tasks with send rights to a port can send messages to that port.

3. A message sent to a port is revealed only to the source and destination tasks.

---

[3]Mach actually allows tasks to share memory with their children but this does not affect the semantics of the IPC abstraction.

[4]Note that we commonly refer to the task holding receive rights to a port as the *receiver* and, correspondingly, to the *sender* or *owner*.

These security guarantees allow an object model of computation, in which ports represent protected objects, to be implemented securely. A user-level task managing a protected object will create a port to represent access to that object, and securely pass send rights for that port to a client. The client can then invoke an operation on the object by sending a message to the port. The Mach kernel guarantees that the port representing the object cannot be fraudulently obtained by a malicious task.

3. Network Communication

By itself, the Mach kernel does not provide any mechanisms to support inter-process communication over the network. However, the definition of Mach IPC allows for communication to be transparently extended by user-level tasks called, by convention, *Network Servers*. Network servers effectively act as local representatives for tasks on remote nodes. Messages destined for ports with remote receivers are actually sent to the local network server as shown in Figure 3-1.
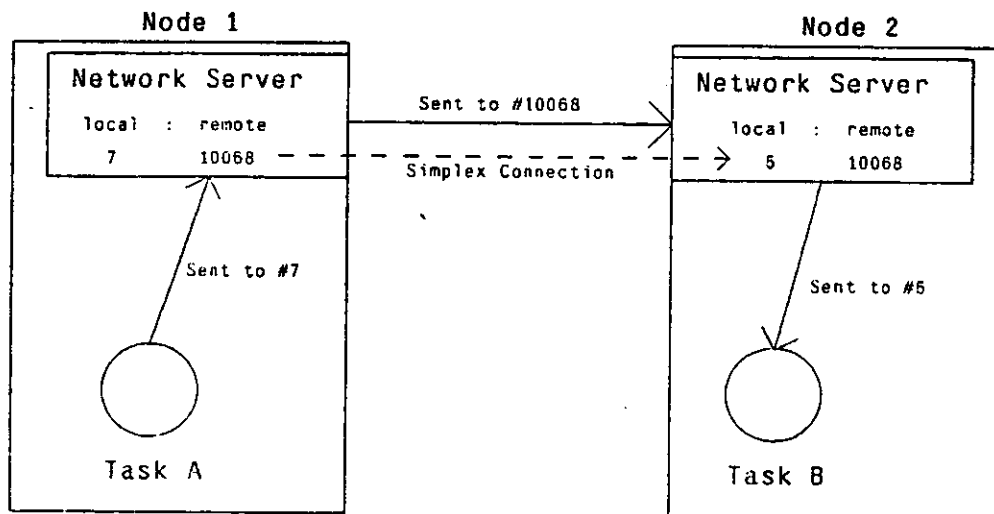
Task A on Node 1 sends a message locally to Port #7. A believes that it is sending the message to task B, but really the message is sent to the local Network Server. This Network Server receives A's message and translates local Port #7 into Network Port #10068. It notes that Network Port #10068 resides on Node 2 so it forwards the message to the Network Server on Node 2. When it receives the message, the Network Server on Node 2 translates Network Port #10068 into local Port #5 and sends the message to this port. The message arrives at its destination which is Task B. (Note that this example is a simplified description of the activity of mapping ports since it omits the mapping that is done by the kernel of each node when a message is sent between the user task and the Network Server.)

Figure 3-1: Sending a Message across the Network

Moreover, when a task sends a message to a destination port on another node, the forwarding of the

message is transparent to the sender. In other words, the sender has no direct means of determining whether the eventual destination port is local to its node or is actually on a remote node.

To support the forwarding of messages between nodes, the network servers collectively implement an abstraction of *Network Ports*. A network port represents a port to which tasks on more than one node have access. Each network port is known by its *Network Port Identifier*. A network server maintains a mapping between those network ports that are accessible to tasks on its node and the corresponding local ports. Thus, for instance, if a local task has access to a port that represents access to a remote server, the network server will maintain a mapping between the local port and the network port that represents access to the remote server. In addition, just as local tasks hold access rights to local ports, so do network servers hold access rights to network ports. In particular, each network server holds receive rights to those network ports for which the receive rights to the corresponding local ports are held by local tasks. Send and ownership rights to network ports are handled in the same way except that send rights to a network port may be held by many network servers.

In operation, when a network server receives a message from a task trying to send a message to a remote destination port, it maps the local destination port into a destination network port identifier. The network server then derives the address of the destination node from the network port identifier and sends the message over the network to this node. The destination network server, on receiving the network message, maps the network port identifier into a local destination port and forwards the message to its ultimate destination. A network server must examine messages sent or received over the network to determine whether they contain port access rights and, if so, update the mapping between local and network ports. Network servers can recognise ports in messages because messages consist of a sequence of typed data objects.

In order to transmit messages between themselves, network servers split messages into datagrams which are transmitted over the network using a network level protocol (currently the DoD Internet Protocol [5]). Each network message is uniquely identified by a combination of the source node identifier and a time-stamp. Datagrams forming one message are numbered in sequence so that they can be reassembled in the correct order by the destination network server, and so that duplicate and lost datagrams can be detected. The actual protocol used is a variation of Richard Watson's *delta-t* connectionless transport protocol [11].

4. Mechanisms for Protecting Network Communication

Malicious nodes can potentially compromise the security of messages and ports in a variety of ways:

- a node can illicitly read messages being transmitted between network servers,
- a node can disrupt or replay the messages sent between network servers,
- a node can fraudulently pretend to have send rights to a network port,
- a node can masquerade as the receiver of a network port and thus receive messages destined for the real receiver, or
- a node can cause another node to believe that a network port has died, thereby depriving that node of access to the port.

Furthermore, ports cannot be used to represent protected access to services and resources unless the underlying network port abstraction is secure.

These threats to the security of messages and ports can be prevented if:

- the data transmitted between network servers can be protected,
- active attacks on message streams can be prevented,
- access rights to network ports can be securely transferred between network servers
- the network server holding receive or ownership rights to a network port can be correctly identified, and
- the death of a network port can be securely detected.

Thus, the network servers must support all of these functions if they are to provide a secure network port abstraction.

4.1. Protecting Message Data

Messages transmitted over the network are protected by encrypting them. We use a secret key encryption scheme, similar to DES [6], in which encryption keys are maintained for each pair of network servers in active communication. Messages sent to the receiver are encrypted by the secret key shared only by the sending node and the receiver's node. These keys are distributed by a trusted, central *Key Distribution Server* using a variation of the Needham and Schroeder key distribution protocol [7].

4.2. Protecting against Active Attacks on Messages

Encryption by itself is not enough to prevent active attack on messages. Such attacks involve the reordering, duplication or modification of datagrams or the replay of network messages. All of these attacks are prevented by the transport level mechanisms provided for both protected and unprotected communication in combination with data encryption. Datagram sequence numbers detect reordering and duplicating. Checksums on datagrams prevent modification attacks. Time-stamps and message sequence numbers can be used to detect message replays.

4.3. Secure Transmission of Network Port Access Rights

Secure exchange of data between network servers does not imply that network port access rights are protected. To protect access rights to network ports, network servers must be able both to transfer the rights between themselves securely and identify a network port's receiver and owner correctly. More specifically, these requirements mean that a network server must be able to:

- authenticate a network server which claims to have send rights to a network port,
- locate the network server currently acting as the receiver for a network port,
- re-locate the new network server which is the receiver for a network port after receive rights to the port have been migrated between nodes, and
- authenticate a network server as the legitimate receiver for a network port.

A network server must also be able to locate and authenticate a network port's owner. In general, the same procedures can be used for locating and authenticating both owners and receivers.

The ability to authenticate a network server claiming access to a port depends on the structure of the network port identifier. A network port identifier consists of:

- a 64-bit *Public Unique Identifier (PUID)* used primarily to locate a new receiver,
- a 64-bit *Secret Identifier (SID)* used to represent send rights,
- the address of the network port's receiver, and
- the address of the network port's owner.

For convenience, a network port identifier is written as:

[PUID, SID, Receiver, Owner]

Each network server is responsible for ensuring that the PUIDS that it generates are unique. Network servers use a combination of their node's address and the current time of day to generate unique PUIDs.

The SID is a random number generated by the network server that created the network port. Only network servers with access to the port know its SID, and therefore knowing the SID is equivalent to possessing send rights to the port. Network servers without send rights to the port are highly unlikely to be able to guess its SID and thus cannot illicitly obtain send access to the port.

Whenever a network server acquires send rights to a network port by receiving a message from another network server, it is important that it be able to identify the location of the port's receiver (and owner). By examining the address of the network port receiver contained in the network port identifier, a sender can identify both a potential receiving node and the encryption key that should be used to protect messages destined for this network port.

Unfortunately, the network server receiving a network port identifier cannot assume that the receiver specified in the identifier is still the current receiver for the port. Since the Mach port abstraction allows

6

for the transfer of receive rights as well as send rights between tasks, receive rights to a port may move from one node to another. When receive rights to a network port move to a different network server, a network server with send access to the port must be able to determine the identity of the new receiver.

The information a sender has about the location of a network port's receiver can be maintained in two different ways: either the information is explicitly updated every time the receiver moves, or it is left up to the sender to obtain new information as necessary. The first alternative requires that a list be kept of all network servers with send rights to a network port. The list could be kept either by a trusted central server, in which case the server must be informed reliably of any movement of access rights, or by the port's receiver, in which case the list must be transferred to a new receiver along with the access rights to the port. Both these approaches involving maintenance of a list of senders are impractical given that the Mach IPC abstraction allows send rights to a port to be transferred not only by the receiver but also by any task with send access to that port. Moreover, the use of a central, trusted server would place an inordinate amount of dependency on a third party. Another alternative is to maintain the list of senders implicitly by using a secure multicast facility to distribute information about the movement of receive rights. It would be up to an individual network server, when it receives send rights to a network port, to join the multicast group for the port. Multicasts would be made secure by encrypting transmissions with a key that is specific to the multicast group. The key would form part of the network port identifier. Such a secure multicast facility, however, is not currently available in our internet environment.

Thus, the only realistic scheme for updating information about a network port's receiver is to place the burden on senders to locate a new receiver. The simplest way for a sender to determine the location of a new receiver securely is to query (and believe) the old receiver. If the original receiver has crashed then the sender can query the network port's owner. However, if both the original receiver and owner have crashed, the sender is left with no direct means of locating the new receiver. In this case the only way in which a new receiver can be located is to make a broadcast[5] request asking if any network server is the receiver for the network port. By definition, broadcasts are not secure and thus no private information can be included in the broadcast request. Hence the network port identifier must contain some identification information that is public and can be used in broadcasts. The PUID described above serves this purpose.

Once a sender has found a potential receiver for a network port, it must verify the authenticity of the newly located receiver. When the identity of the new receiver is obtained from the owner or the old receiver, then the sender trusts the information it has received and believes that the new receiver is

---

[5]The broadcast at the network server level will probably translate into a multicast at the physical network level so that the transmission is restricted to those nodes running a Mach network server.

authentic. On the other hand, when the new receiver's identity is obtained from a response to a broadcast request, then the sender must authenticate the new receiver explicitly. In order to be able to perform this authentication, the sender must have previously obtained a token, that represents the receiver's authenticity, from the original receiver. This token consists of a random number, X, and the network port's secret identifier (SID), both encrypted by KR, a key known only to the receiver (and the owner) and used to represent receive (and ownership) rights. The complete token, as held by a sender, is thus[6]:

$$(X, [X, SID]^{KR})$$

To authenticate the new receiver the sender sends it the token. Only if the new receiver knows KR, and thus is able to decrypt the token and return the random number X to the sender, is it authentic.

4.4. Detecting Network Port Death

A network port dies when both receive and ownership rights to the corresponding local port are deallocated. The local network server is informed by the Mach kernel of the port's death. The network server changes the status of the corresponding network port to dead. In addition, it unreliably and insecurely informs other network servers of the port's death by broadcasting a *port-death* message that names the port's PUID. A sender must be able to confirm the network port's death, whether it receives the broadcast port-death message or not, so that a malicious node cannot take advantage of the port's death.

The location and authentication procedure used for identifying a new receiver for a network port is also used for determining that a network port is dead. The dead port's receiver and owner are queried to determine the dead port's state if they still have information about the port. Otherwise, a sender must resort to a broadcast request for information about the network port. Any responses received as a result of a broadcast must be authenticated as described above. The port can be declared dead if no authentic information about its receiver or owner is received.

5. Protocol Details

A detailed examination of the network server protocols will give more insight into the mechanisms needed to extend the port abstraction securely over the network. The operations crucial to maintaining security are:

- the transfer of send rights to network ports,
- the transfer of receive rights to network ports, and
- the procedure followed by a network server when it needs to determine the existence and location of a network port's receiver or owner.

---

[6][X]^K stands for encryption of X by key K.

5.1. Transfer of Send Rights

Send rights to a network port can be obtained either directly from the network server which holds the receive rights to the port or indirectly from a network server holding send rights to the port. In the first case, shown in Figure 5-1[7], the receiver includes a token along with the network port identifier.

Receiver --> Sender   [PUID, SID, Receiver, Owner, X, [X, SID]^{KR}]

PUID is the network port's public unique identifier. SID is the secret identifier. X is a random number made up by the receiver. KR is the key representing receive (and ownership) rights.

Figure 5-1: Transfer of Send Rights from Receiver

In the second case, shown in Figure 5-2, the network server acquiring send rights to a network port explicitly requests a token from the port's receiver.

Sender$_1$ --> Sender$_2$ Send Rights to [PUID, SID, Receiver, Owner]

Sender$_2$ --> Receiver   GetToken (SID)

Receiver --> Sender$_2$ New token for SID is (X, [SID, X]^{KR})

PUID is the network port's public unique identifier. SID is the secret identifier. X is a random number made up by the receiver. KR is the key representing receive (and ownership) rights.

Figure 5-2: Transfer of Send Rights from a Sender

In both cases, the receiver must create a token, by generating a new random number and encrypting it with the key that represents receive rights. The token can later be used by the sender to authenticate a different network server that claims to be a new receiver for this network port. A sender cannot masquerade as a new receiver because each sender obtains a distinct token from the receiver.

Note that a network server, once it has obtained send access to a network port, retains send access to that network port as long as the port remains alive, even if all tasks on that node with send access have terminated. This is because the Mach port abstraction does not allow a port's receiver to determine

---

[7]Note that all the messages in the protocols are encrypted by the network server to network server connection keys unless specified otherwise.

whether or not any tasks still hold send rights to the port.

5.2. Transfer of Receive Rights

Receive rights to network ports must be transferred *reliably* between network servers if they are to be transferred *securely*. The new receiver must acknowledge that it has received the rights from the old receiver. The network message acknowledgement can act implicitly as the acknowledgement of the transfer when the access rights are transferred explicitly in a message from a task on one node to a task on another node. The new receiver must acknowledge the transfer explicitly when the rights are being transferred because the local task holding the rights either dies or deallocates the port. In addition, in both cases the network server transferring the rights unreliably informs the port's owner about the transfer if the ownership rights are not held by the receiver.

Figure 5-3 describes the protocol for transferring receive rights. Ownership rights, and receive and ownership rights combined, are transferred similarly.

Old Receiver	**Block reception of messages on port**
Old Receiver --> New Receiver	**[PUID, SID, Old Receiver, Owner, KR]**
New Receiver	**Change Network Port Identifier to [PUID, SID, New Receiver, Owner]**
New Receiver --> Old Receiver	**Rights successfully received.**
Old Receiver	**Change Network Port Identifier to [PUID, SID, New Receiver, Owner]**
Old Receiver	**Unblock port**
Old Receiver --> Owner	**Network port is now [PUID, SID, New Receiver, Owner, KR]**

PUID is the network port's public unique identifier. SID is the secret identifier. KR is the key representing receive (and ownership) rights. The message to the owner is sent only if the owner is a different network server from the old or new receiver.

Figure 5-3: Transfer of Receive Rights

A transfer of receive rights to a network port requires synchronisation between senders and receivers if messages are to remain correctly ordered. The network port is blocked during the transfer to ensure that senders do not become confused about the current identity of the receiver. Blocking the port means that the old receiver refuses to accept messages destined for this port but does, however, send back negative acknowledgements to the sender to reassure the sender that the port is still alive. The old receiver unblocks the network port when it is certain that the receive rights are now held by the new receiver. When a sender, that was blocked during the transfer of rights, attempts to resend the message to the old

receiver, it is informed by the old receiver that the receive rights have moved to the new receiver. It is not necessary to block and unblock a network port when ownership rights to the port are transferred.

5.3. Searching for a Network Port

A network server initiates a search for a network port whenever it needs to verify or correct its information about that port. The search consists of the network server sending a number of *port request* messages to other network servers until either it has obtained the desired information or it decides that no network server has any reliable information about the port. We describe the procedure used by a sender to locate a receiver; the procedure to locate an owner is identical.

5.3.1. Port Search Procedure

The first step in locating a network port's receiver is to query the network server that is currently believed to be the receiver. The answer to this request may be any of the following:

- *port here* - the receive rights are still held by the responding network server.
- *port dead* - the responding network server is sure that the network port is dead (for instance, because it deallocated the port). Note that in order for this information to be available, the receiver must keep some information about this network port for some time after the port's death (see Section 6.2 for further discussion about this issue).
- *port not here, transferred* - the responding network server believes that the receive rights are now held by the network server named in the reply.
- *port not here, unknown* - no information about this network port.
- *no response* - the network server is dead.

Note that the answer given by the queried network server is believed because it has or had receive rights to this network port and is still trusted with respect to this port.[8] If the answer is either *port here* or *port dead* then the search concludes successfully. If the *port not here* answer designates another network server, then the searching network server proceeds to query this new network server by restarting the port search procedure. This second network server is trusted because it was named by the old receiver.

If the queried network server provides no useful information about the network port, then the search procedure continues first by querying the port's owner if the owner is not the same as the receiver. The responses from the owner are treated in the same manner as the responses from the receiver.

Finally, if no useful information can be gained either from the network port's receiver or owner, then the searching network server must resort to an unencrypted broadcast *port request* naming the port's

---

[8]In general, once a network server has held receive rights for a port it can never be prevented from acting as a transparent intermediary between tasks with send access and a new receiver. A network server can set itself up as an intermediary by creating a new port, passing receive rights to this new port instead of receive rights for the original port and then intercepting and retransmitting messages sent to the original port.

Public UID. Ideally the broadcast should be reliable, in other words all network servers should receive it and have a chance to respond to it. We simulate a reliable broadcast by repeating the broadcast request a small number of times. Only the real receiver or owner should respond to such a broadcast request. The authenticity of a responding receiver or owner can be verified by using the token that was previously obtained from the original receiver. The authentication protocol is shown in Figure 5-4.

Sender --> New Receiver   Authenticate [PUID, $[X, SID]^{KR}$]

New Receiver   Decrypt $[X, SID]^{KR}$ and check SID

New Receiver --> Sender   [X]

PUID is the network port's public unique identifier. SID is the secret identifier. X is a random number made up by the port's original receiver and known to the sender. KR is the key representing receive (and ownership) rights.

Figure 5-4: Authentication of New Receiver using Token

Only if the new receiver replies with the random number X can the new receiver be trusted. Note that the new receiver only replies to the sender if the correct secret identifier is included in the token.

If the searching network server receives no response to the broadcast request, then the port can be declared dead and deallocated locally.

5.3.2. Implicit Transfer of Receive or Ownership Rights

As well as being used by senders to check up on the status of a network port, the port search procedure is also used by a network port's receiver or owner when the receive and ownership rights to the port are split between two network servers. The receiver or owner periodically checks the status of the complementary network server, using the port search procedure, to determine if the complementary rights have been implicitly transferred to it due to the death of the other network server. For instance, upon the crash of the network server holding receive rights to a network port, the receive rights are transferred implicitly to the network server holding ownership rights to the port.

A network server may effectively hold both receive and ownership rights to a network port but not know it until the death of the complementary network server is detected. This implies that the results of a port request sent out by a sender may be different before and after the receiver or owner has done a checkup. To ensure that a sender does not erroneously decide that a network port is dead, its port search must not fail as long as one of the owner or receiver still responds albeit with incorrect information.

5.4. Case Analysis of Protocols

The security of the network server protocols can be demonstrated by analysing all the events that a network server experiences with respect to one network port. These events result from the activities in which a network server may be engaged. The activities are:

- The transport of IPC messages over the network.

- The transfer of access rights to network ports either explicitly in a network message or implicitly because receive and ownership rights are split between two network servers and one of them deallocates its access rights to the port.

- The unreliable notification of a network server holding one of receive or ownership rights to a network port, that the complementary rights have been transferred to a third network server.

- The unreliable broadcast of a hint about a network port's death hint when both the port's receive and ownership rights have been deallocated.

- The broadcast of a restart message as part of a network server's initialisation; this message informs other network servers that access rights to network ports held by the restarted network server before it crashed are now effectively deallocated.

- The sending of port checkup requests to network servers that may hold receive or ownership rights to a network port; these requests help determine whether the port is still alive or whether its receiver or owner have moved.

In the following sections, we examine the events resulting from these activities in terms of what effect they have when a network server holds no rights, send rights or at least one of receive and ownership rights to a network port. In each case, those events not considered do not affect the security of the protocols.

5.4.1. No Rights

The events that a network server experiences and the actions that it takes when it holds no rights to a network port are:

1. *Reception of send rights.*
   The network server obtains a token of the receiver's authenticity if such a token was not included with the rights just received.

2. *Reception of receive and/or ownership rights.*
   The network server must believe that it really has been given the rights.

Additionally, in both the above cases the network server creates a new local port and establishes a mapping between the local port and the network port. Note that send rights to a port are normally obtained as part of a secure higher level protocol, for example from a trusted name server, thus ensuring that the port's initial receiver can be trusted. A network server that later claims to be a new receiver for this port can be authenticated using the token obtained from the initial receiver.

5.4.2. Send Rights

The relevant events that a network server experiences and the corresponding actions that it takes when it holds send rights to a network port, apart from those arising as part of a locally initiated port search procedure, are:

1. *A request from a local process to send a message to this port.*

   The network server attempts to send the message to the machine currently believed to be the receiver for the network port. If the information about the receiver turns out to be incorrect, then the old receiver will send a negative reply (or no reply if it is dead). The sender must then begin the port search procedure.

2. *Initiation of a regular port checkup.*

   The network server begins the port search procedure. Normally the search will return immediately with a confirmation that the current information is still valid. However, if the reply is negative, or if there is no reply, the search must be continued as described in Section 5.3.

3. *Reception of a broadcast message indicating that the port is dead.*

   The network server begins the port search procedure for the network port suspected to be dead. If no network server answers, or if some trusted server answers that the network port is dead, then the information held about the port can be deleted and the corresponding local port can be deallocated. This deallocation will cause the Mach kernel to send a message to the local tasks that have send access to the port, informing them that the port is now dead.

4. *Reception of a broadcast message indicating that the network server holding ownership or receive rights to the port has restarted.*

   As above, the network server begins the port search procedure for the network port. The network port is either dead or its receive or ownership rights are held by a different network server.

5. *Reception of a port-request that is part of a port search by another network server.*

   In this case, the decision to reply depends on whether or not the network server has relevant information to provide, and whether or not an acknowledgement of the request is expected. If the request is broadcast, then no acknowledgement is expected and the request can be ignored; some other network server having receive or ownership rights will give a more useful answer. If the request is not broadcast, it must be acknowledged. The network server must check that the request is valid by examining the secret identifier, and then must send the information that it currently holds about the network port. Note that in this latter case, the network server replies because it may once have held receive or ownership rights to the network port, and thus its answer will be trusted and used by the querying network server.

6. *Reception of a network message destined for this port.*

   As in the previous case, the network server replies with the information it currently holds about the network port because it must have once held receive rights to the port. The sender will treat the response as a negative acknowledgement and start a port search procedure.

7. *Reception of send rights.*

   The network server checks that the information received about the network port matches the information it currently holds. If there is a mismatch then the port search procedure must be

initiated to determine whether the network port received is indeed the same port as the one already known about. If the mismatch cannot be resolved then the two network ports are treated as separate ports.

8. *Reception of receive and/or ownership rights.*
In this case, the network server will receive the key (KR) that represents receive and ownership rights to the port. The network server can verify that the network server sending the rights is to be trusted by checking that the key is correct. Note that the network server can only do this if it previously has obtained a token of the receiver's (or owner's) authenticity. The key is valid if it can be used to correctly decrypt the token.

5.4.3. Receive or Ownership Rights

A network server holding receive or ownership rights must handle any of the events experienced by a network server holding send rights alone. Additional events that a network server experiences, and the corresponding actions that it takes, are:

1. *Reception of a network message from another network server.*
The network server accepts the message if it holds receive rights to the network port. Otherwise it replies with a negative acknowledgement and gives the information it has about the port's status.

2. *Deallocation of the corresponding local port by a task on the network server's node.*
The network server transmits a broadcast message indicating that the network port is dead if it holds both receive and ownership rights to the port. On the other hand, if the network server holds only one of receive and ownership rights, then the rights are transferred to the network server holding the complementary rights. A local port search procedure will be initiated if the other network server no longer has the complementary rights.

3. *Request from a sender for a token.*
The network server generates a new random number, encrypts it and the network port's SID with the key (KR) that represents receive and ownership rights to the port. It then returns both the token and the new random number to the sender making the request.

4. *Reception of an authentication request.*
The authentication request names the network port's PUID and includes a token. The token is decrypted using the key (KR) that represents receive and ownership rights to the port. If the decrypted token contains the correct SID, then the network server sends a reply that includes the random number obtained from the token.

5. *Request from a local task, that really holds receive or ownership rights to the local port, to transfer one of the rights to a remote task.*
The network server transfers the access rights as part of the normal network message protocol. In addition, if the rights complementary to those being transferred are held by another network server, then this other server is unreliably informed about the transfer of rights.

6. *Reception of the complementary access rights.*
In this case, the network server receives the key (KR) representing receive or ownership rights to the network port. To check that the network server sending the rights is to be trusted, the key

received in the message must be compared with the key currently associated with the network port.

7. *Reception of a broadcast indicating that the machine believed to be holding the complementary rights has been restarted.*

   The network server initiates a port search procedure for each of the network ports that may have been lost. Even if the broadcast message contains incorrect or fraudulent information, the port search procedure will always determine the correct information.

8. *Reception of a hint indicating that the complementary rights have been transferred to a third network server.*

   In this case, the transfer hint contains the key (KR) representing receive or ownership rights to the network port. Before updating the information kept about the network port, the network server checks the validity of the transfer hint by comparing the key received in the hint and the key currently associated with the network port.

6. Discussion

6.1. The Cost of Security

Security never comes for free. In making Mach's network communication secure, the main added cost comes from having to encrypt message data. Other additional costs come from the protocols needed to protect access rights to network ports when they are transferred between nodes and from the procedures required to locate and authenticate new receivers for network ports.

One approach to dealing with the cost of security is to ensure that the protection mechanisms are only used when necessary and not by default. Secure and insecure messages should be distinguished and treated separately. The extra work required to protect message data and securely transfer port access rights need only be done for secure messages. Any ports that appear in a secure message are treated in a secure fashion, and have their access rights transferred securely over the network. In addition, any port that is being treated as secure should never be included in an insecure network message.

Another approach to reducing the cost of security is to take into account which actions are common and which are rare. Common actions should not cost as much as less frequent actions, even when performed securely. For example, send rights to network ports are far more frequently transferred between network servers than are receive or ownership rights. Thus the cost of securely transferring send rights across the network should be kept low, even if in doing so the cost of transferring receive or ownership rights is increased.

Reducing the cost of transferring send rights is of particular importance when considering remote procedure call interactions. In a remote procedure call, send rights to a reply port are transferred on every

call. Thus, if, as described in Section 5.1, tokens are to be used to authenticate receivers, a new token needs to be generated on every call. To prevent this unnecessary expense, the reply port could be treated as a special case and not have a token associated with it. Alternatively the receiver could keep a list of those network servers to which it has already sent tokens. The cost of keeping such a list can be made relatively small, because the list does not have to be totally accurate, and it is not necessary to transfer this list when receive rights to the network port move to another network server.

Two further schemes for reducing the cost of securely transferring send rights do not use a token at all. In the first scheme, the receive and ownership rights to the network port would be split between two different network servers before any send rights to the port are transferred. A sender could then authenticate a new receiver by simply querying the owner, provided that the owner maintains up-to-date information about the identity of the receiver. In the second scheme, public key encryption would be used to authenticate a network port's receiver. The network port identifier would contain a public key, and the corresponding private key would be known only to the receiver and owner. To authenticate a receiver, a sender would encrypt a random number with the public key and send the result to the receiver. Only if the receiver knows the private key could it correctly decrypt the random number, increment it and return it, still encrypted by the public key, to the sender.

6.2. Implementation Issues

When implementing the network server protocols, there are several design decisions to be made that influence the behaviour of the protocols but do not affect their security.

There are tradeoffs involved in the determination of when a port checkup should be initiated and how exactly a port search should proceed. A port checkup could be triggered every time new or conflicting information is received about a network port. However the resulting port search is wasted if no use of the port is made before more new or conflicting information is received. On the other hand message transmission latency would be increased if the port search were always delayed until strictly necessary.

The port search itself can proceed either by querying individual network servers or by resorting to broadcasts early in the search procedure. Using individual messages may unnecessarily prolong the search procedure. Using broadcasts will waste processing time at network servers that have no information about the network port.

In addition, it is useful to distinguish between *port checkups* and *port requests*. A port checkup occurs when a network server is just making sure a network port is alive and asks another network server for information about the port. A port request occurs when a network server thinks that it has incorrect information about a network port and wants to update it. In particular, a port request received by an

owner or receiver should trigger the owner or receiver into checking its own information. The port request is an indication that the information held by one of the sender, receiver or owner is incorrect.

Another issue concerns the length of time for which a network server needs to maintain information about a dead network port. A network server holding both receive and ownership rights to a network port declares the port dead when both access rights are deallocated locally. The network server then broadcasts a death hint naming the port's public unique identifier. The broadcast hint prompts network servers with send access to the dead network port into confirming the death of the port. Broadcasting a hint has the advantage of reducing the period for which the sender falsely believes that the network is still alive. It has the disadvantage, however, of potentially causing the receiver to be swamped with requests for information about the port. If no broadcast is made, senders can still find out about the death of the port as part of their regular check up on the port's status. In either case, the receiver has to maintain information about the dead network port until no more status requests arrive. To determine when this information can be discarded, a time-out is set, and, after the time-out expires, the network server can assume that all senders have found out about the death of the network port. The time-out can be set to be a simple function of the checkup interval if all network servers regularly check up on all the ports to which they have access. Even if some network server still does not have up-to-date information after the time-out has expired, it can later broadcast an information request and determine the fate of the port securely.

Lastly, it is still open to debate whether, when receive and ownership rights to a network port are split, both the receiver and owner should always maintain accurate information. Maintaining accurate information has advantages. For instance, the cost of the port search procedure could be reduced if the owner of a network port always has reliable information about the port's receiver. To ensure that both the receiver and owner of a network port hold the same information about the port implies that a two phase commit protocol must be used whenever receive or ownership rights are transferred. However, the expense of the commit protocol is not justified if the holder of the complementary rights does not use the information about the transfer. Currently we have decided to inform the complementary network server unreliably. This may be more expensive than the reliable transfer if a network server is often forced to resort to the port search procedure to determine the new location of the complementary rights, but is cheaper when network communication is highly reliable.

7. Conclusions

We have shown that it is feasible to extend a capability based system securely over a network. Such a scheme has a number of important advantages over end-to-end encryption as the basis for building a secure distributed system:

- The number of user-level protection abstractions is reduced by relying solely on the port-based capability system for all protected access to services and resources.
- All secure information that must be transmitted over the network is protected by default rather than relying on haphazard encryption which may or may not be done by user processes.
- Encryption is avoided between tasks on the same node and may also be avoidable when the physical network itself is known to be secure.

The Mach network server currently operates on VAX architecture machines within the CMU Department of Computer Science (including MicroVAX I & II, VAX 11/750's, 11/780's and 11/785's). It does not yet (March 1986) perform encryption, but we are in the process of integrating the protocols discussed in this paper into the network server.

Acknowledgements

References

[1]   G. Almes and G. Robertson.
An Extensible File System for Hydra.
In *Proceedings of the 3rd International Conference on Software Engineering*. IEEE, May, 1978.

[2]   Robert Victor Baron, Richard F. Rashid, Ellen H. Siegel, Avadis Tevanian, Jr. and Michael
Wayne Young.
MACH-1: A Multiprocessor Oriented Operating System and Environment.
In *New Computing Environments: Parallel, Vector and Systolic*, pages 80-99. SIAM, 1986.

[3]   F. Baskett, J.H. Howard and J.T. Montague.
Task communication in DEMOS.
In *Proceedings of the Sixth ACM Symposium on Operating Systems Principles*. November, 1977.

[4]   David R. Cheriton and Willy Zwaenepoel.
The Distributed V Kernel and its Performance for Diskless Workstations.
In *Proceedings of the Ninth ACM Symposium on Operating Systems Principles*. October, 1983.

[5]   National Technical Information Service ADA079730.
DOD Standard Internet Protocol.
Technical Report IEN-128, Defence Advance Research Projects Agency, January, 1980.

[6]   National Bureau of Standards.
Data Encryption Standard.
Federal Information Processing Standards Publication 46, U.S. Department of Commerce, 1977.

[7]   R.M. Needham and M.D. Schroeder.
Using Encryption for Authentication in large Networks of Computers.
Communications of the ACM 21(12):993-999, December, 1978.

[8]   Richard F. Rashid and George G. Robertson.
Accent: A Communication Oriented Network Operating System Kernel.
In *Proceedings of the Eighth ACM Symposium on Operating Systems Principles*. December,
1981.

[9]   Jerome H. Saltzer and Michael D. Schroeder.
The Protection of Information in Computer Systems.
Proceedings of the IEEE 63(9):1278-1308, September, 1975.

[10]   Robert D. Sansom.
Security in a Network Operating System.
In *Securicom 86 - 4th Worldwide Congress on Computer and Communications Security and
Protection*. March, 1986.

[11]   Richard W. Watson.
Timer-Based Mechanisms in Reliable Transport Protocol Connection Management.
Computer Networks :47-59, February, 1981.