Distributed communication services in the Masix system

Franck Mével and Julien Simon
Laboratoire MASI, Institut Blaise Pascal
Université Paris VI
4 place Jussieu
75252 PARIS CEDEX 05, FRANCE
Tél: (1) 44 27 71 04
Fax: (1) 44 27 62 86
{Franck.Mevel,Julien.Simon}@masi.ibp.fr
http://www-masi.ibp.fr/masix

Abstract

A current trend in operating system research consists in implementing the system in user mode with a minimal kernel. This kernel, called a microkernel, provides resource management and some elementary services, e.g. physical memory management or process management. Many operating systems, both centralized and distributed, have been built on top of a microkernel. Nevertheless, none of them has really investigated the problems related to multiple personality support, that is, running simultaneously on the same workstation applications from worlds as different as Unix, DOS, or OS/2.

Masix is a distributed multi-server operating system based on the Mach microkernel, with multiple personality support. Its main feature is a *Distributed Generic Layer* (DGL), which offers distributed services to the personalities. The distributed multi-server architecture of Masix grants it a high modularity, but also raises many issues, such as transparency, security and performance, which cannot be solved without adequate communication services.

To provide total transparency, we extend the traditional Mach communication model to a workstation network by interposing a *Generic Network Server* (GNS) between the tasks and the microkernel. We defined a global name service, based on a name resolution protocol, which allows any pair of Mach remote tasks to communicate transparently, using the local Mach IPC semantics. Our name server also provides local and remote authentication mechanisms, based on digital signatures and a secret key algorithm. To prevent eavesdropping, all remote communications are transparently encrypted by the GNS, using a public key algorithm. These security measures can be easily merged into the name service, to yield a secure distributed name resolution protocol.

Microkernel based-systems are traditionnaly criticized for their relatively poor performance. As far as network services are concerned, experiments show that a good performance level can be reached, provided that the distinctive features of microkernels are taken into account.

1 Introduction

In the past few years, the design of distributed operating systems has been moving towards a microkernel approach. Indeed, microkernels offer many advantages compared with monolithic systems, such as portability, modularity and dynamicity. However, microkernel-based systems raise new issues, related to the scattering of information. To solve them efficiently, the operating system must have appropriate ditributed communication services at its disposal.

Masix [1] is a distributed operating system based on the Mach microkernel [2], currently under development at the MASI Laboratory. After a quick overview of the communication services available in Mach, of the structure of Masix and of related work, we study some of the requirements of the Masix system as far as communications are concerned, i.e. transparency, security and good performance. Then, we present the solutions we have designed.

2 The Mach microkernel

Masix is based on the Mach 3.0 microkernel, designed at Carnegie Mellon University. Mach has been widely used to build many Unix-like systems, centralized like Lites [3] and Mach-US [4], or distributed such as Sprite [5]. Other operating system personalities have been implemented on top of Mach, like DOS [6] or OS/2 [7]. As far as we know, no distributed system has investigated the problems related to multiple personality support.

Mach offers powerful inter-task communication mechanisms [8], based on ports. A port is a queue, in which messages can be added or removed. Actions on ports are carried out via rights, which are granted to the tasks. A task can own three types of rights on a port :

1. a receive right, granted to a single task ;
2. a send right, granted to several tasks ;
3. a send-once right, allowing the task to send one — and one only — message on the port.

Each port is managed by a single task, which owns the receive right on the port. Some privileged ports are managed directly by the kernel itself. Ports are accessed by the tasks via port names (numeric identifiers), which are internally converted into rights by the kernel. A task can send or receive messages on a port. A message is a structure containing data, made of :

- a header, describing the message : size, name of the source port, name of the destination port, message type, . . . ;
- a set of typed data : type, number of objects, value of the objects.

3 Structure of the Masix system

The primary goal of Masix is the simultaneous execution of multiple personalities, in order to run concurrently on a same workstation applications from the Unix, DOS, OS/2 and Win32 worlds. Masix also pools the resources of a workstation local area network (LAN), independently from the personalities that run on each node, thanks to a Distributed Generic Layer (DGL).

Masix is built according to the multi-server model, which grants it a high modularity. Indeed, each server can be written and tested independently from the others. Also, it is possible to add or remove dynamically a server to the system. As shown on Figure 1, Masix is made up of two layers, themselves built of several servers :

- the personalities, which implement the semantics of traditional operating systems : Unix, DOS, OS/2, Win32, . . . ;
- underneath, the DGL, which offers generic services to the multiple personalities (communication, shared memory, process management, fault tolerance, ...

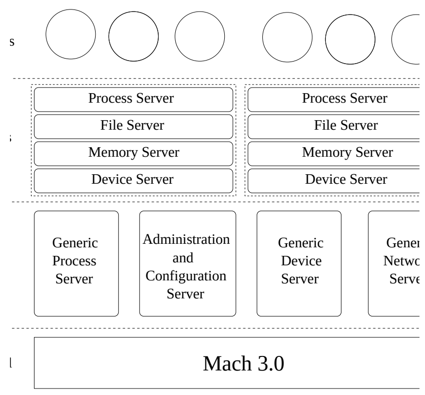

Figure 1: Structure of the Masix system

4 Requirements

Many issues arise because of the modularity of Masix, linked to the distribution of the global state among the microkernel and the various servers. As a consequence, many communications are initiated between servers running on a same node on the one hand, and between servers running on different nodes on the other hand.

We study three of these issues :

- transparency,
- security,
- performance.

4.1 Transparency

To simplify the design and implementation of Masix, it is desireable that remote IPCs follow the same semantics as the local IPCs provided by Mach. Thus, inter-task communications are based on the **mach_msg()** primitive, no matter where the tasks are located. This way, two communicating tasks do not have to make assumptions about their respective locations. This is all the more important since some mechanisms of Masix such as migration, presented in [9], can move a running task from one node to another.

4.2 Security

In Masix, traditional network security issues take a new dimension. The most important are eavesdropping, message tampering, replaying and masquerading.

Eavesdropping Communication links are vulnerable to physical probes, which listen to the network traffic. If such an attack was successful, the confidentiality of inter-node communication would be compromised. It is extremely difficult to prevent these attacks because the network can hardly be physically secure. Nevertheless, they can be defeated by systematically encrypting every message sent on the network. This way, even if a message is eavesdropped, its content is unintellegible ;

Message tampering An ill-disposed or buggy task could intercept a message, alter it and send it again. For example, it could change its destination, or modify the data it contains. However, if the messages are properly encrypted, they cannot be altered without the destination task noticing it. Furthermore, this type of attack seems difficult to set up on an broadcast type of network, such as Ethernet, and is probably better suited to a store-and-forward type of network ;

Replaying old messages A spy could replay a sequence of messages that it eavesdropped, which would alter the state of the system. This attack cannot be defeated by encryption alone, since the attacker does not have to understand the messages. As a consequence, timestamps must be used to prevent an old message from being processed again ;

Masquerading A process could gain access to otherwise protected information by usurping the identity of an authorized process. Thus, a server could be lured by a fake client, which would use its services without authorization, and consequently alter its state. In the same manner, a client could be lured by a fake server, and disclose confidential information. As a consequence, communicating entities must mutually authenticate themselves before any message exchange takes place.

4.3 Performance

Most measures indicate that microkernel-based systems have worse networking performance than monolithic systems [10]. One might then conclude that microkernels are inherently unfit for distributed architectures. Yet, most of the systems based on the Mach microkernel rely on unappropriate networking code, because it has been written for monolithic systems. For example, the CMU single-server system, Unix Server [11], runs code written for 4.3BSD. As we shall see later, a microkernel-based system can match the networking performance of a monolithic system, provided that the distinctive features of the microkernel are taken into account.

5 Related work

We present two major projects related to Mach network IPC : the netmsgserver and NORMA.

5.1 Netmsgserver

The netmsgserver, designed at Carnegie Mellon University [12], was the first project whose goal was to transparently extend the Mach local IPC to a workstation network.

It is made of a multi-threaded Mach task, running in user mode on every node of the network. The netmsgservers exchange information, so that each of them may have a coherent view of the tasks running on every node.

The main services provided by the netmsgserver are :

- port translation, thanks to a database containing the following data :
  - a local port, representing a local task ;
  - a network port, tagged by a unique identifier ;
  - information related to the security of the communication.
- port management, i.e. checking the validity of the database, and updating it if necessary ;
- transport protocol management, i.e. segmentation and reassembly, flow control, error detection and recovery ;
- message conversion : some information included in traditional Mach messages is meaningless to a remote task, such as port rights. Thus, it has to be converted before the message is transmitted, then converted back when it is received. Converting the data included in the message is also mandatory when the two communicating entities do not share the same data internal representation (little-endian or big-endian) ;
- name resolution ;
- message encryption, with an algorithm derived from DES [13] ;

Thus, the netmsgserver offers transparent and secure communication services to Mach tasks. Unfortunately, its performance level is moderate, because each message operation requires costly context switches between the task, the server and the kernel.

5.2 NORMA

NORMA, designed by the Open Software Foudation [14, 15], is an extension of the Mach microkernel, which allows remote tasks to communicate with the semantics of traditional local Mach IPC. This way, remote communication is totally transparent.

When a task gives a send right on one of its ports to a remote task, NORMA takes charge of this port : it becomes thus a NORMA port. Each one of them is tagged by a unique identifier. Each node manages a lookup table, containing the NORMA ports it is aware of.

NORMA offers transparent communication services, but requires extensive modifications to the microkernel. However, to the best of our knowledge, it doesn't include authentication or encryption services. Thus, it is probably better suited to tightly-coupled multiprocessor computers than to workstation networks.

6 The Generic Network Server

The Generic Network Server (GNS) offers communication services capable of suiting the requirements of every component of the Masix system, i.e. the generic servers and the personality servers.

6.1 Transparency

To achieve total transparency, we extend the traditional Mach communication model by interposing the GNS between the tasks and the microkernel.

The first step of every communication, local or remote, is name resolution. In Masix, name resolution is provided by the Administration and Configuration Server (ACS). Our name resolution protocol enables a task to request from its local ACS the name of every task running on the network.

Name resolution protocol Let us consider two tasks A and B, running on different nodes, registered with their respective name servers. We assume that neither A nor B has ever communicated with a remote task. We call ACS_a, ACS_b the name servers and GNS_a, GNS_b the network servers. The protocol works as shown on figure 2 :

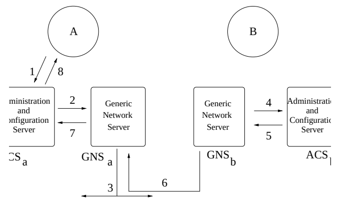
A
B
1
8
Administration and Configuration Server
2
7
Generic Network Server
Generic Network Server
4
5
Administration and Configuration Server
ACS a
GNS a
GNS b
ACS b
3
6

Figure 2: Name resolution between two remote tasks

1. A sends B's name to ACS_a, and requests the corresponding port ;

2. since B is a remote task, ACS_a doesn't know this port. So, it requests the resolution of B's name from GNS_a ;

3. GNS_a broadcasts a name resolution request, containing the name of the wanted task, as well the node number of the request's originator ;

4. GNS_b gets a copy of the request. Since B has never communicated with a remote task, it is unknown to GNS_b. So, GNS_b sends B's name to ACS_b and requests the corresponding port ;

5. ACS_b sends B's port to GNS_b ;

6. GNS_b stores this information for further use, and answers to the request of GNS_a by sending it B's port and its node number. The pair *(node number,port)* constitutes a global identifier, thanks to which a task can be located.

7. GNS_a stores this identifier, and creates a proxy port for B, giving itself the receive right. Then, it sends the proxy's name to ACS_a, and gives it a send right on this port ;

8. ACS_a sends the proxy's name to A, and gives it a send right on this port. As soon as A gets this information, it can communicate with B, as shown on figure 3 ;

9. A sends a message on B's proxy port ;

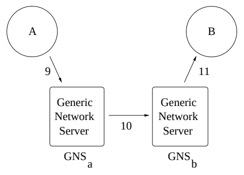

Figure 3: Communication between two remote tasks

10. since GNS_a owns the receive right on this port, it receives the message. It translates the local port names (B's proxy port and A's reply port) into global identifiers, and sends the message to GNS_b ;

11. GNS_b accomplishes the inverse translation and delivers the message to B.

Optimizations This name resolution protocol may seem extremely time-consuming. However, the full resolution of B's name only takes place once on A's node. Let us suppose that another task running on A's node wishes to communicate with B. It asks ACS_a for B's port (step 1). Since this information is cached by ACS_a, the task gets an immediate answer (step 8). Anyhow, if a task located on a third node wishes to communicate with B, the full resolution of B's name must occur on that node. Nevertheless, further resolutions of B's name can be prevented by modifying step 6. Instead of replying only to GNS_a, GNS_b can broadcast B's unique identifier to every GNS on the network. Thus, B will be known by every node, and the resolution of its name will only consist in a call to the local GNS.

What happens if B wants to reply to A ? First of all, A's name must be resolved in the same way B's name has been. Again, this full resolution can be prevented by modifying step 3. Instead of sending A's global identifier only to GNS_b, GNS_a broadcast it to to every GNS on the network. This way, A will also be known by every node.

These optimisations induce extra remote communication between the nodes. However, insofar as unicasts are replaced by broadcasts, the number of messages does not increase. As a consequence, we believe that the cost of this extra communication will be very low. We are currently implementing this protocol, and extensive performance measurements will determine its efficiency.

6.2 Security

Our goal is to provide the environments and the DGL with services allowing them to defeat the attacks we mentioned earlier, or at the very least to complicate their implementation.

Our approach is based on the following principles :

- separation of the inter-server and the client-server communication interfaces ;
- encryption of every message sent between the network servers, thanks to a public-key algorithm [16] ;
- authentication of the servers by the ACS ;
  - when they register, thanks to digital signatures ;
  - when they request a name resolution, thanks to a secret key algorithm ;
- mutual authentication of the servers, thanks to a secret key algoritm ;

Communication interface separation

Each server has two communication interfaces :

- one through which it receives the clients' requests. Its ports are available on request from the ACS ;
- another one through which it communicates with other servers. Its ports are also available on request from the ACS, but their delivery is submitted to authentication. Prior to any port delivery, the ACS makes sure that the other party is a legitimate server.

This separation protects the ports used by inter-server communication, and prevents a rogue process from acquiring a send right on one of them.

Message encryption

Encryption in Masix is based on a public key algorithm. This method does not require any beforehand knowledge of a secret key shared by the communicating entities. This is why it is well suited to distributed architectures.

Principle The principle of such an algorithm is very simple. Each communicating entity is attributed two keys : a public key and a secret key. Both are generated by a mathematic algorithm, such as RSA [17] or LUC [18], which guarantees that it is extremely difficult to compute the secret key from the public key. The public key of an entity is used to encrypt the data sent to this entity. So, before being able to send a message, one must obtain the public key of the addressee. When the addresse receives the message, it decrypts it with its secret key, which must be unknown to anybody but its owner.

Public key distribution Each GNS knows the public key of all the others. Indeed, when a GNS starts, it broadcasts its public key to its homologues, and requests that they send it theirs. This exchange must of course be encrypted and authenticated, to prevent a user process from masquerading as a GNS. One solution to this problem could be to encrypt the request with the super-user's key, whose ownership acts as an authentication.

Encryption Let us consider two tasks A and B, which want to communicate in a secure manner. We note $C_K(...)$ a message encrypted with key K.

A secure communication takes place as follows :

1. A sends a message on B's proxy port ;
2. GNS_a receives the message, because it has the receive right on this port. It then encrypts the message with the public key of GNS_b, pub_{R_b}, and sends it to GNS_b ;
3. only GNS_b can decrypt it, thanks to its secret key, sec_{R_b}. Once this is done, it delivers the message to B ;
4. when B replies to A, GNS_b encrypts the message with pub_{R_a}, and sends it to GNS_a ;
5. only GNS_a can decrypt it, thanks to sec_{R_a}. Once this is done, it delivers the message to A ;
6. and so on ...

Server authentication by the ACS

At start-up, the personality servers and the DGL servers register with the ACS. However, a user process must at all costs be prevented from doing so. If it succeeded, it could request the ports of all the "private" server ports, which would allow it to obtain privileged information, or to alter the system's state.

When it wishes to register, a server must send a port to the ACS, as well as the name under which the port should be registered. The ACS asks the process server to find the owner of the port, and to compute the digital signature of the owner's binary. This computation could for instance be based on the MD5 algorithm [19].

If this signature is not listed in the signatures authorized by the ACS, this means that the process which tried to register is not a legitimate server (the list of signatures could for instance be stored as a disk file, encrypted with the super-user's key). In this case, the registration fails.

If the signature is listed, the registration succeeds, and the ACS sends back to the server a signature, which now acts as a secret key. The ownership of this key proves that its holder is a legitimate server. Thanks to this key, which must be provided at each name resolution request, the ACS can make sure that its party is a registered server and not a impostor.

Authentication of local servers

We use a secret key authentication protocol, inspired by [20]. The signature provided by the ACS when a server registers acts as the secret key of the server. Authentication between two servers A and B, illustrated in figure 4, takes place as follows :

1. A sends to ACS_a the message (A, B), i.e. A's and B's identities ;

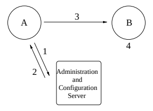

Figure 4: Authentication betwen two local servers

2. ACS_a computes a new secret key, sec_{ab}, based on sec_a and sec_b, which will only be used for communications between A and B. Then, it sends to A the message $(B, sec_{ab}, C_{sec_b}(sec_{ab}, A))$;

3. A stores sec_{ab} for further use. Then, it sends to B the message $C_{sec_b}(sec_{ab}, A)$;

4. only B can decrypt this message and extract sec_{ab}. From now on, only A et B have a copy of this key. As a consequence, its ownership guarantees the identity of its holder.

The original protocol includes an extra message exchange. Indeed, there is no guarantee that the message received by B on step 4 comes from A. It could be an eavesdropped message, replayed by a process which does not have sec_{ab}. However, since this is a local communication, it cannot be eavesdropped.

Mutual authentication of remote servers

When A and B run on different nodes, the distribution of the secret key sec_{ab} is more complicated, as shown on figure 5 :

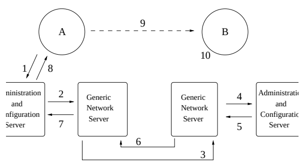

Figure 5: Authentication between two remote servers

1. A sends to ACS_a the message (A, B) ;

2. ACS_a sends to ACS_b the message (A, B, sec_a). Since ACS_b is remote, GNS_a receives the message on its proxy port ;

3. GNS_a sends to GNS_b the message $C_{pub_{GNS_b}}(A, B, sec_a)$;

4. GNS_b decrypts it with sec_{GNS_b}, and sends to ACS_b the message (A, B, sec_a) ;

5. ACS_b computes a new secret key, sec_{ab}, based on sec_a and sec_b, which will only be used for communications between A and B. Then, it sends to ACS_a the message $(A, B, sec_{ab}, C_{sec_b}(sec_{ab}, A))$. Since ACS_a is remote, GNS_b receives the message on its proxy port ;

6. GNS_b sends to GNS_a the message $C_{pub_{GNS_a}}((A, B, sec_{ab}, C_{sec_b}($

7. GNS_a decrypts it with sec_{GNS_a}, and sends to ACS_a the message $(A, B, sec_{ab}, C_{sec_b}(sec_{ab}, A))$;

8. ACS_a sends to A $(A, B, sec_{ab}, C_{sec_b}(sec_{ab}, A))$;

9. A stores sec_{ab}. Then, it sends to B the message $C_{sec_b}(sec_{ab}, A)$;

10. only B can decrypt this message and extract sec_{ab}. From now on, only A et B have a copy of this key. As a consequence, its ownership guarantees the identity of its holder.

This authentication protocol is extremely similar to the name resolution protocol. They can consequently be merged easily, so that localization and authentication happen simultaneously.

6.3 Performance

A microkernel-based system can match, or even exceed, the networking performance of a monolithic system, thanks to the following techniques :

- taking into account the distinctive features of the microkernel, such as :
  - mapping the device driver into the applications' address space [21] ;
  - sharing memory between the network server and the device driver [22] ;
- improving the internals of the protocol, without modifying its original semantics ;
- preventing the network-related system calls from being processed by a Unix emulator, and rewriting network applications so that they directly call Mach primitives. This can be easily done for traditional applications, such as **ftp** or **telnet**, but not for every user application ;

The most notable optimization has been presented in [23]. Indeed, it enables the system to reach the networking performance of a monolithic system. This is achieved by splitting a protocol functionalities between the network server

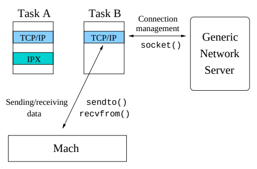

Figure 6: Network protocol decomposition

and a library mapped into the applications' address space. This decomposition is shown on figure 6.

The performance-critical services, such as sending or receiving a message, are located in the library. This way, many costly context switches between the application and the network server can be prevented. The server only contains the services which require its explicit intervention, such as opening or closing a connection. We are currently working on this decomposition.

7 Conclusion

The distributed multi-server architecture of the Masix system requires appropriate communication services. They should be transparent, secure and performant. Our solution is based on a Generic Network Server, interposed between the personalities and the microkernel. By using proxy ports and a global name resolution protocol, we scaled the local Mach IPC up to a workstation network, without altering their original semantics. We also proposed several mechanisms which guarantee the confidentiality and authenticity of the communications. They rely on secret and public key cryptographic algorithms. In particular, we have presented a distributed authentication protocol, which establishes the identity of two remote tasks. This protocol can easily be merged with the name resolution protocol. Lastly, we studied several techniques, which improve the end-to-end networking performance. The most notable is based on the decomposition of network protocols, so that performance-critical primitives are located in the applications' address space.

References

[1] R. Card, E. Commelin, S. Dayras, and F. Mével. The MASIX Multi-Server Operating System. In *OSF Workshop on Microkernel Technology for Distributed Systems*, June 1993.

[2] M. Accetta, R. Baron, W. Bolosky, D. Golub, R. Rashid, A. Tevanian, and M. Young. Mach: A New Kernel Foundation For UNIX Development. In *Proceedings of the Usenix 1986 Summer Conference*, June 1986.

[3] J. Helander. Unix under Mach - the Lites Server. Master's thesis, Helsinki University of Technology, December 1994.

[4] J. Stevenson and D. Julin. Mach-Us: Unix on Generic OS Object Servers. In *Proceedings of the 1995 Usenix Conference*, January 1995.

[5] Michael D. Kupfer. Sprite on Mach. In *Proceedings of the Third Usenix Mach Symposium*, April 1993.

[6] G. Malan, R. Rashid, D. Golub, and R. Baron. DOS as a Mach 3.0 Application. In *Proceedings of the 1st Usenix Mach Symposium*, pages 27–40. Carnegie Mellon University, November 1991.

[7] J. Phelan, J. Arendt, and G. Ormsby. An OS/2 Personality on Mach. In *Proceedings of the 2nd Usenix Mach Symposium*, pages 191–201, April 1993.

[8] R. Draves. A Revised IPC Interface. In *Proceedings of the Usenix Mach Workshop*, pages 101–121, October 1990.

[9] F. Mével and J. Simon. Building a Distributed Generic Layer for Multiple Personality Support on top of the Mach Microkernel. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, November 1995.

[10] C. Maeda and B. Bershad. Networking Performance for Microkernels. In *Proceedings of the 3rd Workshop on Workstation Operating Systems*, April 1992.

[11] D. Golub, R. Dean, A. Forin, and R. Rashid. Unix as an Application Program. In *Proceedings of the Summer 1990 USENIX Conference*, pages 87–96, June 1990.

[12] R. Sansom, D. Julin, and R. Rashid. Extending a Capability Based System into a Network Environment. In *Proceedings of ACM SIGCOMM'86*, pages 265–274, 1986.

[13] NBS. Data Encryption Standard (DES). In *Federal Information Processing Standards 46*. US National Bureau of Standards, 1977.

[14] B. Bryant, A. Langerman, S. Sears, and D. Black. Norma IPC: A Task-to-Task Communication System for Multicomputer Systems. Technical report, Open Software Foundation, October 1993.

[15] OSF. Norma IPC Version Two: Architecture and Design. Technical report, Open Software Foundation, October 1994.

[16] W. Diffie and M. Hellman. New Direction in Cryptography. *IEEE Transactions on Information Theory*, IT-22:644–654, 1976.

[17] R. Rivest, A. Shamir, and L. Adelman. A method of obtaining digital signatures and public key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[18] P. Smith. LUC Public Key Encryption. *Dr Dobb's Journal*, 1993.

[19] R. Rivest. The MD5 Message-Digest Algorithm. Technical Report RFC 1321, Network Information Center, 1992.

[20] R. Needham and M. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *Communications of the ACM*, 21(12):993–999, 1978.

[21] A. Forin, D. Golub, and B. Bershad. An I/O System for Mach 3.0. In *Proceedings of the 1st Usenix Mach Symposium*, pages 163–176, November 1991.

[22] F. Reynolds and J. Heller. Kernel Support For Network Protocol Servers. In *Proceedings of the Usenix Mach Symposium*, November 1991.

[23] C. Maeda and B. Bershad. Protocol Service Decomposition for High Performance Networking. In *Proceedings of the 14th ACM Symposium on Operating Systems Principles*, pages 244–255, December 1993.