Overview of the Architecture of Distributed Trusted Mach

E. John Sebes
Trusted Information Systems
444 Castro Street, Suite 800
Mountain View, CA 94041
ejs@ba.tis.com

Abstract

The Distributed Trusted Mach (DTMach) Concept Exploration produced a design that extends Trusted Mach (TMach), which is TIS's development of a B3 trusted version of the Mach operating system. The overall goal of DTMach is to integrate distributed systems functionality with the security mechanisms of TMach. TMach's modifications to the interprocess communications mechanisms are combined with transparent network communications based on similar functionality from Mach. In addition, DTMach defines extensions to the TMach system servers that enable them to utilize this distributed IPC to provide their services in a distributed manner.

Introduction

Distributed Trusted Mach (DTMach)[1] incorporates the previous work of Mach and Trusted Mach (TMach)[3][4][5], and extends them to provide the basis for a trusted distributed operating system designed to meet the B3 security requirements[6]. This paper provides a summary of the DTMach architecture and key design features.

The term "distributed system" has many definitions, but we take the working definition of a system in which: the presence of separate nodes in the system is largely invisible; system operations have the same functional behavior regardless of which node they are invoked from; and system resources are managed on a system-wide basis.

The Mach system was designed to support distribution, in that system resources can be utilized from any of several Mach nodes connected by a network. However, Mach is not a distributed system in the above sense. This is due to the fact that when system resources are not local to a user's node, the user must frequently take this fact into account when accessing those resources. In other words, Mach system software does not yet take full advantage of Mach features to make system service fully transparent and uniform across nodes in a network.

TMach, on the other hand, is a *stand-alone* trusted system that is derived from and compatible with Mach. TMach does provide for the use of network devices, and allows for untrusted implementations of network protocols. However, it is beyond the current scope of the TMach development to support the trusted communication between TMach nodes that would support the extension of TMach system services throughout a networked system. Therefore, TMach does not address trust enhancement for the Mach functionality that supports distribution.

The purpose of DTMach, therefore, is two-fold. First, DTMach must extend TMach to provide trust enhancements for the same sort of distribution support that Mach provides. Secondly, DTMach must build on these extensions to actually provide, in a trustworthy manner, the distribution of services that Mach itself does not provide.

To achieve the purposes described above, DTMach combines TMach trusted IPC with Mach's distributed IPC, and adds extensions to ensure the security of the combined trusted distributed IPC. In keeping with the multi-server architecture, this functionality is provided not by kernel modifications, but by a server called the Network Server. This service provides the basis for trusted networked virtual memory management, which, together with the extended IPC, forms the foundation on which trusted distributed system servers can be built. DTMach defines a general approach to a distributed system service, designed to allow for the extension of TMach system servers without requiring extensive redesign. Using this approach, each of the TMach system servers will be extended to allow its service to be provided in a distributed manner.

DTMach Architecture

The software components of DTMach fall into three main layers:

- The TMach kernel;
- The new DTMach components: the Network Server and the Network Pager;
- The TMach system servers, extended to provide distributed service.

The TMach kernel is used in DTMach, and needs no additional functionality to handle the extension of kernel services into a distributed environment. Instead the Network Server and Network Pager tasks provide the distribution of kernel services — IPC and memory management, respectively. The only change required to the TMach kernel is an additional privilege needed by the Network Server, described below. This privilege can be provided by the existing TMach kernel privilege mechanism.

The Network Server is the DTMach component that provides the functionality of trusted distributed inter-process communication (IPC). The TMach kernel provides trusted IPC, but only on a single machine. In Mach, the kernel's IPC is extended in a distributed manner by the Mach NetMessage Server, but this IPC lacks the security features of TMach. The DTMach Network Server combines and builds on both of these areas of previous work. As a result, DTMach IPC is the same as TMach IPC, but is distributed in a manner similar to that of Mach[7][8].

Just as the Network Server extends TMach IPC in a distributed manner, the Network Pager extends TMach virtual memory management so that the same services are available in a distributed environment. TMach memory management is built on the same basic functionality of ports and messages that constitute IPC. As a result, the Network Pager uses the trusted distributed IPC provided by the Network Server.

Both the Network Server and Network Pager extend kernel functionality in a way that is invisible to the users of kernel services. For example, when sending an IPC message, a client task cannot determine whether its final destination is on the same node– with the IPC service provided by the kernel– or on another node, with the service extended by the actions of the Network Server. In general, the involvement of the Network Server and Network Pager in providing these "kernel-level" services is invisible to clients. As a result, we say that the Network Server and Network Pager operate at the "kernel layer" of the system. That is, they provide the same services as the kernel, but in a distributed manner. However, they can do so without having to execute in the same privileged state and hardware-protected address space in which the kernel executes.

While the TMach kernel provides the building blocks of operating system services, the bulk of operating system services are provided by the TMach system servers. The same is true of DTMach. However, in DTMach, the TMach system servers are extended so that each server on each node not only provides its service on that node, but also co-operates with the same server on other nodes, in order to provide the service in a distributed manner. For example, the TMach File Server provides services for the use of files that are physically located on the node that the File Server runs on. In DTMach, the File Server also does so, but also communicates with other File Servers on other nodes. As a result, a client can contact the File Server on its node, and request service for any file without regard for which node the file is physically located on; if the file is on a remote node, the local File Server can arrange for the service request to be forwarded to the appropriate remote File Server. The basis of this cooperation and communication is the trusted distributed IPC provided by the Network Server, and distributed VM provided by the Network Pager.

Network Server

The primary function of the Network Server is to act as intermediary and message transport facility in IPC transactions where the sender and receiver are on different nodes in a DTMach system. The overall approach to message handling is similar to that in Mach [7][8], as is the approach to communication between the various instances of the Network Server on the various nodes in a system. However, the IPC mechanism supported is that of TMach, and an essential requirement of the DTMach Network Server is to enforce relevant aspects of the TMach security policy.

Network Server Architecture

The DTMach Network Server is composed of three separate parts, each of which is implemented in a separate task:

Net Message Server: is similar in overall functionality to the Mach Net Message Server, but has several significant additional security-related requirements.

Net Protocol Server: implements the various network communication protocols used by the Net Message Server.

Net Line Server: manages the network devices.

The essential reason for this breakdown is that it allows the Net Protocol Server to be an untrusted component. That is, the trusted Net Message and Net Line Servers do not rely on the secure operation of the Net Protocol Server. Therefore, the Net Protocol Server can not violate any aspects of the TMach security policy. The Net Message Server does have security-related requirements, including the assurance that security-relevant data about each message (e.g. the label and identity of the sender) is accurately sent with each message. The Net Line Server also has security-related requirements; the network devices are multi-level devices, and must be managed by a trusted component to ensure that the devices are used in accordance with the TMach security policy.

This arrangement is made possible by certain protection measures enforced by the Net Message and Net Line Servers on the Net Protocol Server. Details of this approach are given in [2]. As a result, existing network communication protocols and protocol implementations can be used in DTMach, without any re-engineering for trust-related functionality or assurance requirements.

The interaction of these three components in one IPC transaction is summarized as follows:

1. The Net Message Server receives a message destined for another node.
2. The Net Message Server determines where and how to send the message, and passes it to the Net Protocol Server.
3. The Net Protocol Server breaks the message up into packets suitable to send over a network, and sends each to the Net Line Server.
4. The Net Line Server applies integrity measures to each packet, and writes it on the network device.
5. The packets arrive on the destination node.
6. The Net Line Server on the destination node reads packets from the network device, checks the integrity of each packet, and passes each intact packet to the Net Protocol Server.
7. The Net Protocol Server on the destination node re-assembles packets into a message, which it passes to the Net Message Server.
8. The Net Message Server on the destination node delivers the message to its destination task.

Network Server Interactions

Each DTMach node has an instance of the Network Server, which has three main kinds of interactions: interactions with clients, with the local kernel, and with remote Network Server instances.

The interactions with clients are based on the port management strategy of the Network Server: on each node, the Network Server instance is the receiver for local ports for which the actual receivers are on other nodes. In other words, the Network Server is the local stand-in for remote tasks to which local tasks can send IPC messages. As a result, the Network Server can both: receive from local clients every message bound for another node; and, send to the proper local clients each message received via the network from a Network Server instance on another node.

The intra-Network-Server interactions center around the transport of IPC messages from one node to another. In addition to exchanging such messages, Network Server instances also exchange control messages that track changes in system-wide IPC data, such as the senders and receivers of networked ports.

Figure 1 shows these two interactions. Task A sends over a port a message intended for another client task. However, that client is on another node. Therefore, the receiver of the port is, unknown to Task A, the local Network Server instance, which sends the message to the Network Server instance on the proper remote node. There, the Network Server sends

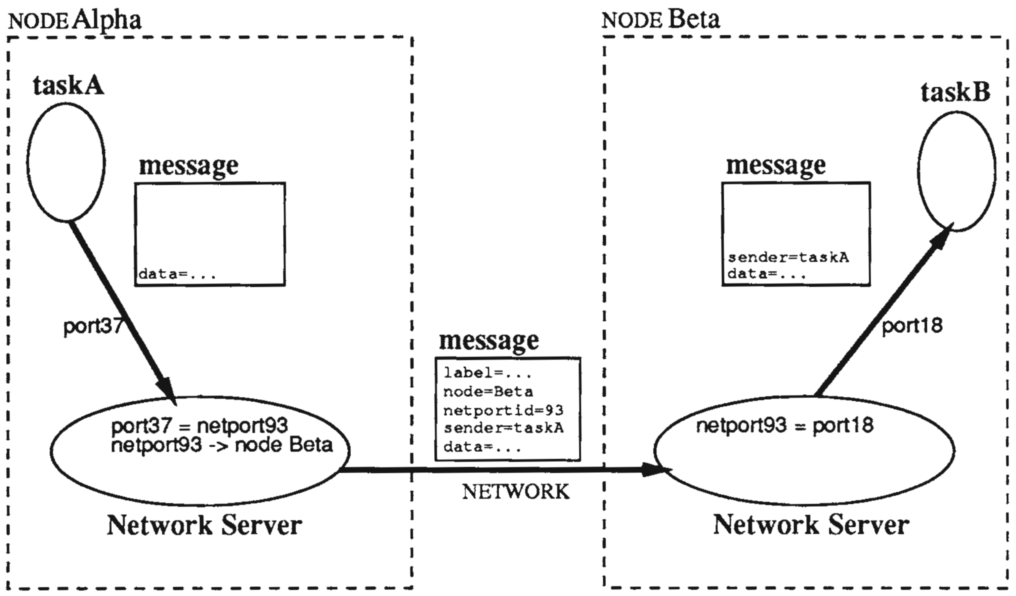

Figure 1: DTMach Message Interactions

the message over a local port to the final receiver of the message. Furthermore, the message send is done in such a way that it appears that the sender was Task A, rather than the Network Server.

The Network Server's interactions with the kernel arise from the kernel's role as the provider of the local IPC service. Therefore, the Network Server does not directly interact with local clients; rather, it uses the kernel's service to send and receive messages from clients. Furthermore, the Network Server requires a privilege from the kernel, which the Network Server uses when it sends a message to a local client in order to deliver it. Normally, the message as received by the client task would indicate that the sender was the Network Server task. Using this privilege, however, the Network Server can tell the kernel to set the message's sender-identity to that of the original sending client task on the remote node.

Figure 1 also shows some of the Network Server's security-related functionality. For each message it moves between nodes, the Network Server adds security-related information to the message — information that is stripped off and used by the receiving Network Server instance. These are schematically shown in the figure as the "label" field of the message in the middle of the figure. In reality, this information includes not only the security label of the message, but also information about the identity of the sender. This information is used on the receiving node to enforce the TMach security policy.

Other Network Server Functionality

In addition to the key message passing and security-related functionality outlined above, there are a variety of other functional requirements of the Network Server:

- Mutual authentication between Network Server instances on different nodes.

- Cryptographic services required for this authentication (both of these are described in more detail in [1]).
- Enforcement of message non-disclosure and integrity protections against incorrect behavior of the untrusted Net Protocol Server ([2]).
- Protection of the Net Protocol Server task, to prevent tampering by clients which could result in denial of service.
- A variety of administrative co-ordination, for example ensuring consistent interpretation of labels across the distributed system.

In all of this functionality, the Network Server assumes that the network is protected, either physically, or by cryptographic devices attached to the network. Lacking physically protected networks, DTMach systems that carry classified information must use government-approved cryptographic devices. In cases of DTMach systems handling non-classified but sensitive information (including commercial environments and non-military government environments) where the network is physically unprotected, the non-disclosure and integrity of network data can be ensured by packet-level encryption carried out by the Net Line Server. The necessary cryptographic services would already be part of the Network Server for purposes of mutual authentication.

Network Pager

The Network Server's IPC service plays a key role in DTMach's extension of the TMach kernel's memory management services into a distributed environment.

As in Mach, memory management in TMach is split between the kernel and tasks called pagers. The External Memory Management Interface (EMMI) defines how the kernel and pagers communicate in order to fill their respective roles in the management of memory objects. Memory objects are kernel-defined objects that represent a region of memory which may be mapped into one or more task's address space. The EMMI also defines security constraints which ensure that operations on memory objects cannot violate the TMach security policy. The kernel enforces these constraints, as well as constraints which prevent untrusted pagers from violating the policy.

Operations on a memory object (by pagers or clients of pagers) are made using operations on the kernel-managed port which represents the object. Since memory objects are represented by memory object ports, and the EMMI is carried out by IPC, it is a simple matter for memory management communication to be carried out over a network. Since the IPC is transparent, so is the memory management communication. Hence, the distributed IPC also serves as a basis for distributed memory management. Thus, in a distributed environment, an external pager may reside on a different node from some of its clients. In this case, the kernel's communication with the pager is just like any other component using distributed IPC. The local port that the kernel sends messages on is connected via the NetMessage Server to the port that the pager receives kernel messages from. The messages and responses are exactly the same as if the external pager were a local process.

In addition to operations on memory objects, TMach memory management also governs the use of memory references (or "out-of-line data") in IPC messages. When a message is sent on a port for which the real receiver is on a remote node, the Network Server acts as an intermediary, as described above. When such a message has a memory reference in it, then

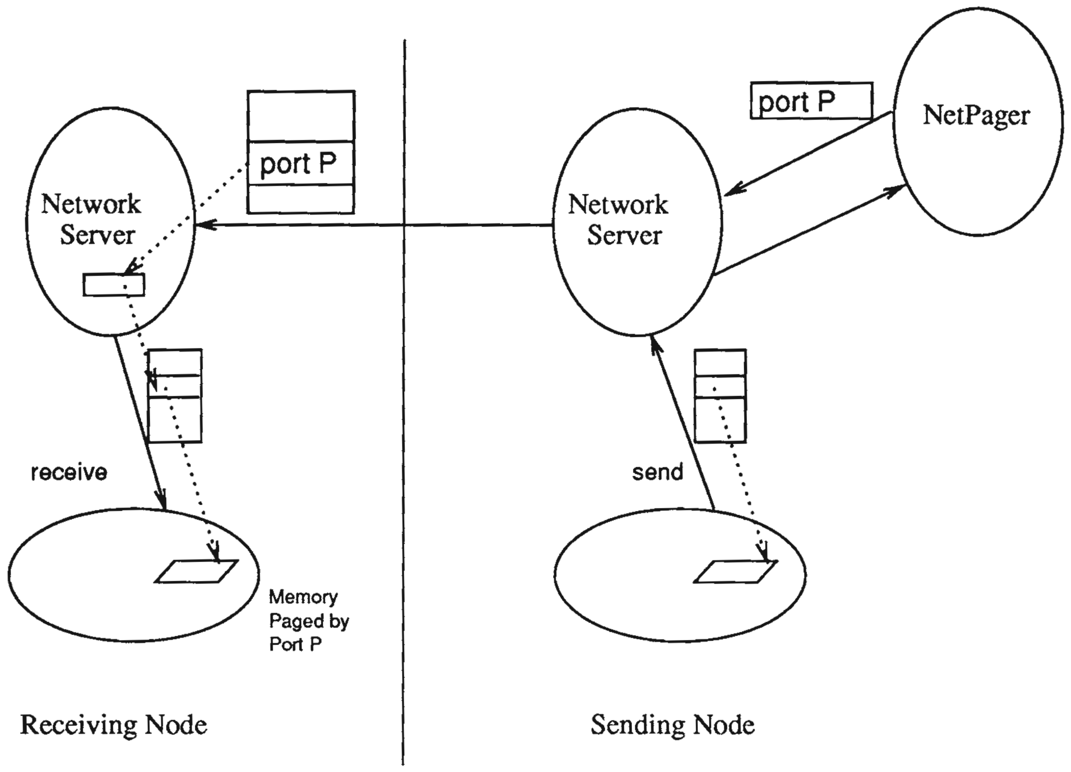

Figure 2: Network Pager Scenario

the Network Server must take action to ensure that the memory reference is meaningful on the destination node. It does so by calling on the Network Pager, the function of which is to manage such out-of-line data by acting as a pager for it.

Each node has a Network Pager task, just as each node has a task that is an instance of the Network Server. The services of a Network Pager task are initiated only by the local Network Server instance, although subsequently the Network Pager communicates with other tasks as part of its paging duties. As a result, the Network Pager can be considered as an "outboard" part of the Network Server, which is implemented as a separate task in order to minimize the duties of the Network Server proper.

The basic interactions between the Network Server and Network Pager are shown in Figure 2. In the figure, the client task in the lower right sends a message over a networked port, and the message has a memory reference in it (indicated by a box with an arrow pointing back to the client's address space). The Network Server receives the message, and notices the memory reference. The Network Server sends to the Network Pager a message containing the memory reference. The Network Pager returns to the Network Server a port representing this memory. The Network Server then takes the original client's message, and constructs from it a message in which the memory reference in replaced by the memory port. This is the message which is transported across the network to the remote Network Server, which delivers it to the destination client task as a memory reference. From then on, the remote client's use of the memory reference proceeds normally, even though the management of the referenced memory is actually carried out in part by the remote Network Pager.

The trust issues related to distributed memory management are largely related to the way the distributed memory management is used by other components, rather than to the details of the mechanisms. That is, the distributed memory management mechanisms described here do not add any new requirements for constraints or checks, other than those already a part of the existing EMMI definition, and the IPC service. For example, a client task may receive virtual memory only from a pager at the same security label, or from a trusted multi-level pager (such as the Network Pager). This rule is enforced by the kernel for local memory management operations; it applies and is enforced just the same in the distributed case as well.

One issue is dependency layering. To meet B3 requirements, the Network Pager (and the Network Server, on which it depends) must be implemented in such a way that they do not depend on any other TCB components which use their services– otherwise, there could be a circular functional dependency which could violate the required architectural layering. This issue does constrain the design of the Network Server and any Network Pagers, but the design in DTMach obviates the difficulty by having these components depend only on the kernel, and not use any of the services of other TCB components.

Covert channels are also a security issue, because pagers are a likely source of covert channel activity. Pagers gauge the paging activity of their clients and the demands on physical memory. Tasks running on the same node as a pager may be able to gauge the pager's activity and infer information about the activity of the clients.

Finally, security labeling is an issue, because the labels of the memory objects passing over a network must be kept consistent across the various nodes in the system. This is handled on two fronts. First, the EMMI definition and the Network Server functionality are sufficient to accurately move the label representation along with memory objects and memory references. Second, administrative measures are necessary to ensure that the same label representation on various nodes actually denotes the same label value on all those nodes.

Servers

In DTMach, as in TMach and Mach, most of the operating services are implemented in servers. DTMach servers build on the existing TMach servers, adding functionality that takes advantage of the distributed IPC and VM services so that servers on different nodes can communicate with one another. As a result, the communicating servers can provide a distributed service.

A distributed server is composed of a number of tasks (which we call server instances) on various nodes. For example, the distributed File Server in a very simple, small DTMach system might be composed of a File Server instance task on each node. Each of these tasks would be similar to the TMach File Server, with the addition of functionality to communicate with other instances, and to propagate shared information.

Approaches to Distributed Services

There is a range of options for expanding TMach system services in a coordinated manner among a collection of networked hosts. In the minimal case, the only distributed server would be the Network Server (which is by definition a distributed server). It would serve to connect the central node providing most services with other "server-less" nodes via distributed IPC. This approach is undesirable because it obviously presents a large problem with the essential distributed system goal of reliability. Also, the approach is untenable at larger scales because the central node is a bottleneck for service.

Another approach is to have all the nodes have a full complement of TMach servers, of which only the various Name Servers communicate with one another. In TMach, the Name Server manages the entire system's name space of objects, and is the focal point for all requests for access to objects. In this approach, there would not be a true distributed Name Server, however. Rather, a system-wide name space would be created using the TMach feature of the "mount point". One central Name Server would be the root of the entire name space, and the other Name Servers on the other nodes would mount their local name spaces onto the central one. The Name Servers would use distributed IPC to forward to one another requests from local clients for remote objects. Other servers would communicate only with local servers. The problem with reliability, while ameliorated somewhat, is still significant. The problem with scaling and performance, while different in some specifics from the previous approach, is also significant. However, this approach is technically workable on a small scale.

Goals for Distributed Services

The deficiencies of approaches such as these lead to the need for fully distributed servers. Such cross-node intra-server cooperation is necessary to provide service that is:

- truly distributed, e.g. largely transparent with respect to the existence of different nodes;
- meets trust requirements;
- meets reasonable goals for distributed system properties.

The goals for DTMach's distributed services take into account both trust requirements, real-world usability, and the limitations imposed by the interactions of the various goals. That is, it would be desirable to build a highly reliable, high-performance, large scale system, but that would be beyond the scope of trying to combine trust with distributed system functionality. Instead, the goals are to create a system which:

- meets B3 trusted system requirements;
- can scale up to approximately 100 nodes (though not larger);
- can respond to some multiple points of failure, (though not to many simultaneous points of failure);
- can operate on interconnected LANs (though not in large internetworked environments);
- can provide very tight consistency among some system-wide databases;
- has production-quality performance.

Server Issues

For each server, the effects of these requirements and goals are somewhat different. In some cases, the effect is the same. For example, reliability is an issue common to all distributed servers, in which each instance of a trusted server will depend on other instances to cooperatively provide service. However, each instance must also be prepared to deal gracefully with system failures, in which other instances may become unavailable. In particular, each instance must continue to operate correctly and securely, independently of its connection to other instances. In some situations, of course, this may mean that the instance provides limited service, or no service at all.

In other cases, different issues will affect different servers quite differently. For example, consistency of server data is a critical issue for servers whose internal data is security-critical, such as authentication data. For these servers, handling system-wide databases must ensure the complete consistency of the data. For other trusted servers which do not have security-critical data, internal database consistency can be maintained more flexibly.

The following provides a summary of how each server design deals with some of the major points.

Name Server is a distributed server, which has an instance on all nodes. Each instance is the central point of contact for object access for all the clients on that node. Much of the co-operation between instances is oriented to maintaining a system-wide name space that is identical throughout the system.

The Name Server provides services for some items in the name space (such as directories); for items of other types (e.g. files), the management is provided by another server. However, for all items, the clients' initial access requests are approved or denied by the Name Server; each approved request is either handled by the Name Server itself (e.g. for directories), or forwarded to another server in cases where the Name Server does not provide service for the item. Additionally, each server instance might not provide service for a particular item, even if the Name Server does provide service for items of that type. In such cases, the instance forwards access requests to the instance that does manage the particular object.

Particular items that are managed by the Name Server can be replicated, i.e. simultaneously managed by more than one instance of the Name Server. For replicated items, read access can be provided locally for each of the nodes whose Name Server replicates the object. However, strict cooperation between replicating server instances is required for modification of replicated items. This cooperation must not only prevent conflicting multiple writes, but also ensure that the updated object data becomes effective at the same time in each participating instance. This is required for the Name Server because the objects that it manages contain security-critical data— such as Access Control Lists of directories– which must not become inconsistent.

Authentication Server is also a distributed server, which typically would have an instance on a relatively small portion of the nodes in a DTMach system. Multiple instances are required, so that authentication service (e.g. login) is reliably available. Since the authentication service should be the same throughout the system, the various instances must maintain a shared authentication database. This is a security-critical database — one that changes, albeit rather slowly, and only under the control of trusted administrative personnel.

Therefore, the main addition to existing TMach Authentication Server functionality is the cooperation required to maintain this database with strict consistency. This approach is required so that the security policy is enforced in the same way throughout the system. However, it has inherent problems at large scales, where it may frequently be the case that all of the Authentication Servers are not available to cooperate on database updates. Successfully dealing with this conflict of interest will require future work in defining manageably-sized administrative domains (in which the scaling problem is avoided) which can inter-operate for authentication (to provide consistent authentication service).

File Server may be a distributed server, with instances on every node that has mass storage devices used to store information useful throughout the system.

However, a distributed File Server is not a strict requirement for a distributed system, which could have separate stand-alone TMach File Servers which only know about the files resident on the local node. Distributed access to files is provided, in any case, by the Name Server, since the Name Server can forward file access requests to the Name Server instance on a node where the file is resident. This Name Server instance forwards accepted file access requests to the local File Server. That local File Server need not be aware of the fact that the request came from another node.

A distributed File Server is a real benefit, however, so that it can provide replication services, and provide more flexible routing of file access requests, relieving the Name Server of the responsibility of correctly determining the location of files.

The main trust issue of the File Server pertains to maintaining system-wide consistency of files which are used by trusted servers to store security-critical data.

Audit Server exists in TMach to provide a repository for all audit data on a TMach system. In DTMach, there is typically an Audit Server on every node in a system. In some cases it is possible for an Audit Server on one node to act as the repository for data for other nodes, but network reliability problems can make it difficult to meet trust requirements for audit. As a result, it is simpler to have an Audit Server on every node.

The Audit Server may or may not be a distributed server. If not, then each system has a separate server, and other means must be used for the required functionality of system-wide aggregation and analysis of audit data. A distributed Audit Server, however, would have an instance on each node. These instances are essentially TMach Audit Servers, with functional additions for communication in support of system-wide aggregation and analysis.

Type Server manages the items that represent the types of other (non-type) items in the name space. For system-defined types, the type data is static. Therefore, there is no need for dynamic consistency cooperation of type data. User-defined type data can be added or changed, but exact consistency is not required for security. Therefore, looser consistency mechanisms could be employed in a distributed Type Server. Since the Name Server and the Type Server work so closely together, it is easiest for the Type Server to be distributed in the same manner as the Name Server.

Other Servers Other TMach servers need not be distributed servers in DTMach. For example, the Device Server manages the allocation of control of local devices to local tasks, and there is no need for a Device Server to know anything about devices on other nodes.

Summary

Distributed Trusted Mach extends the IPC and VM services of the TMach kernel, so that the services can be provided in a distributed environment. This extension requires no functional changes to the kernel, since the distribution functionality is provided by tasks called the Network Server and Network Pager. These services provide the basis for the distribution of the rest of the system services, which are implemented in tasks called servers. TMach system servers provide these services in a manner consistent with B3 trust requirements. In DTMach, some of these servers are extended to be distributed servers. These distributed servers have instances on several nodes, and these instances cooperate to provide services in a distributed manner. The resulting design promises a system which combines trust features and distributed system functionality, while meeting moderate distributed systems goals and stringent trust requirements.

Acknowledgments

The work described in this paper resulted from the efforts of: L. Badger, M. Bernstein, R. Feiertag, N. Kelem, W. Morrison, H. Orman, D. Rothnie, E.J. Sebes, G. Skinner.

References

[1] Trusted Information Systems, "Distributed Trusted Mach Concept Exploration Final Report," Rome Air Development Center, 1990. TIS Rep. 374.

[2] E. John Sebes and Richard J. Feiertag, "Trusted Distributed Computing: Using Untrusted Network Protocols," *Proceedings of the 14th National Computer Security Conference*, October 1991.

[3] M. Branstad, H. Tajalli, F. Mayer, "Security Issues of the Trusted Mach System," Rep. 138, Trusted Information Systems, January 1988.

[4] M. Branstad, H. Tajalli, "Security Policy for the Trusted Mach Kernel," Rep. 179, Trusted Information Systems, September 1988.

[5] M. Branstad, H. Tajalli, F. Mayer, D. Dalva, J. Graham, "Access Mediation in Trusted Mach," Rep. 203, Trusted Information Systems, March 1989.

[6] National Computer Security Center, "Trusted Computer System Evaluation Criteria," DoD 5200.28.STD, December 1985.

[7] R. D. Sansom, *et al*, "Extending a Capability Based System into a Distributed Environment," *Communication of the ACM*, February 1986.

[8] R. D. Sansom, "Building a Secure Distributed Computer System," Carnegie-Mellon University Rep. CMU-CS-88-141, 1988.