JFS: A Secure Distributed File System for Network Computers

Marcus O’Connell, Paddy Nixon
Department of Computer Science, Trinity College Dublin
email: {Marcus.OConnell, Paddy.Nixon}@cs.tcd.ie, http://www.cs.tcd.ie/

ABSTRACT

Network-centric computing aims to solve the problems associated with traditional client/server systems, namely the high performance requirements, and costly maintenance of, the computing resources. With the network-centric model, applications are stored on servers and downloaded to clients as required, where they are then executed locally. User data is saved back to the remote server - local user data is only temporary.
However these applications lack a standard file system where they can store their data files. For instance, when these applications are run as Java applets in a web browser, they are prevented from accessing the local storage system by the Java environment. This paper proposes the Java File System (JFS), a distributed file system for use by these applications, which provides strong security and client/server heterogeneity.

1. Introduction

In computing environments today there is a shift away from the conventional client/server paradigm, to a more network-centric environment [18]. In the client/server model, user data is centralised on servers, but applications reside and execute on “fat client” desktop computers, which have local storage and full operating system. While these desktop machines provide good performance with friendly user interfaces, they are expensive in terms of their high performance requirements and maintenance is costly, as it must be performed locally. The network-centric model tries to solve the problems associated with these conventional client/server systems. With the network-centric model, applications are stored on servers and downloaded to clients as required, where they are then executed locally. User data is saved back to the remote server.

However these applications lack a standard file system where they can store their data files. When these applications are run as applets in a web browser, they are prevented from accessing the local storage system by the Java environment. This paper proposes the Java File System (JFS), a secure distributed file system, implemented in Java, for use by these applications. JFS provides both a strong security model and client/server heterogeneity in both hardware and operating system.

The rest of this paper is organised as follows: Section 2 outlines the requirements for a heterogeneous distributed file system. Section 3 provides a background to distributed file systems, followed by a description of several distributed file system implementations. Section 4 discusses the security issues in distributed systems and some possible solutions to these issues. Section 5 presents the design and implementation of the JFS, including the file server, security architecture and the concurrency control mechanisms.

2. Design Requirements

There are a number of requirements that need to be addressed in the design of a heterogeneous distributed file system. In particular, these are in the areas of distribution, security and concurrency control.

Distribution: JFS should allow the distribution of the file service across a number of servers. In particular, the following distributed file system requirements should be addressed;

Access transparency – JFS clients should be unaware of the distribution of files across multiple JFS servers.

Location transparency- JFS clients should see a uniform file name space, which is independent of the server where the files are located.

Hardware and operating system heterogeneity – JFS should provide a mechanism to allow multiple file servers, each running on a different file system type, to interact with each other, thus providing one global file system to its clients.

Security: Due to the inherent insecurity of the internet, which is vulnerable to both eavesdropping and identity falsification, there is a need to both secure all communication and verify the identity of all parties in a secure manner. The security architecture of JFS should provide authentication of both client and server, in addition to securing all communication between them.

Concurrency: As the file server may be accessed by multiple clients, the issues of concurrent access to files needs to be addressed. The Java File System should provide a locking mechanism, which allows both exclusive and shared access to files.

3. Distributed File Systems

A distributed file system "provides a framework in which access to files is permitted regardless of their locations" [23]. A distributed file system typically consists of a *file service*, which is the specification of the file system's interface, and a *file server*, which implements all or a part of the file service. A distributed file system may have more than one file server contributing to the file service, each located on a different machine. This distribution of file servers should remain transparent to clients.

File services can be split into two types, depending on whether they support an *upload/download* model or a *remote access* model [22]. In the upload/download model, whole files are transferred between client and server. The client carries out operations on the file locally, before uploading the modified file to the server. The advantage of this model is its conceptual simplicity. There is no complicated file service interface, and furthermore, whole file transfer is highly efficient. However, sufficient storage must be available on the client to store all the files required. Furthermore, if only a fraction of a file is needed, moving the entire file is wasteful. In the remote access model however, the file service provides a large number of operations for manipulating files on the server. In this way the file system runs on the servers, not on the clients. This had the advantage of not requiring a large amount of space on the clients, and eliminates the need to download entire files when only small pieces are needed.

File servers may be categorised into two types; *stateless* or *stateful*. With a stateless server, the parameters passed to each procedure call contain all the information necessary to complete the call, and the server does not have to keep track of any past requests. This makes the recovery of a crashed server very easy. A second advantage of stateless servers is efficiency. There is no limit on the number of open files on the server as no information needs to be maintained about the clients that currently have the files open. If state is maintained on the server (i.e. stateful) however, recovery is much harder, and both the client and server need to reliably detect crashes. Performance is however the main advantage of stateful servers. As the state of each client is stored, read ahead of files is possible, and duplicate messages from clients can be detected and ignored.

3.1 Network File System (NFS)

The Network File System (NFS), developed by Sun Microsystems, was designed as a file access protocol, to provide transparent, remote access to file systems [16]. Each NFS server exports one or more of its directories, which clients can ‘mount’ onto a directory in the local file system. NFS defines two client-server protocols. The first NFS protocol is responsible for mounting. The second NFS protocol allows directory and file access to clients.

3.2 Andrew File System (AFS)

The Andrew File System is a distributed file system developed at Carnegie Mellon University. The design of the Andrew File System reflects an intention to support information-sharing on a large scale [4]. The entire contents of files are transmitted to client computers by AFS servers. These files are then stored in a client cache, which contains several hundred of the files most recently used. This cache is permanent, surviving reboots. Local copies of files are used to satisfy clients’ open requests in preference to remote copies whenever possible.

3.3 WebNFS

WebNFS was developed by Sun Microsystems as an extension to the NFS server in order to eliminate the connection overhead [1,2,21]. WebNFS also makes information on NFS servers available to web browsers and Java applets. In particular, the main drawbacks of the NFS that WebNFS solves are:

- First implementation used UDP only.
- Portmapping requirement awkward for firewalls and proxies.
- Separate mount protocol requirement.

WebNFS servers can also be accessed through a URL, for example nfs://server:port/path. This enables World Wide Web browsers to browse documents on WebNFS servers. This also allows Java applets to access files stored on the servers.

3.4 Transparent Cryptographic File System (TCFS)

The TCFS was developed as an extension to the NFS in order to provide a robust security mechanism at the lowest possible cost to the user [3]. One of the major flaws with NFS is that it assumes a very strong trust model. The user trusts the remote file system server (which might be running on a machine in a different location) and trusts a potentially insecure network with sensitive data. Impersonification of users is also a security drawback of NFS. Most of the permission checking is performed in the kernel of the client. In such a context, an intruder can temporarily assign to his own workstation the Internet address of the victim. Without secure Remote Procedure Calls (RPC), no further authentication procedure is requested.

The security mechanism of the TCFS must guarantee that secure files should not be readable by:

- users other that the legitimate owner;
- tapping the communication lines between the user and the remote file system server;
- the super user of the file system server.

In addition to the above, it is also required that all sensitive data should be hidden. This means that for each file not only the content but also the filename is encrypted.

The security of the file contents is guaranteed by means of the Data Encryption Standard (DES) [17]. A feature of TCFS is that users need not remember the decryption password associated with the files. This allows the use of random and difficult to guess passwords.

The implementation of TCFS is based on a daemon which includes code to handle the RPC generated by the kernel and the management of extended attributes. The RPC is syntactically compatible to the NFS RPC but the one relative to read, write and directory handling have been extended to perform security operations. Each time a new file handle is created, the extended attribute "secure" is tested. If the file is secure, then all successive read and write operations will be filtered through the encryption/decryption layer. If the bit is not set, the daemon will behave exactly as an NFS daemon.

4. Distributed Security Systems

Distributed systems are more vulnerable to security breaches than centralised systems as the system is comprised of potentially untrustworthy machines communicating over an insecure network [8]. The following threats to a distributed system need to be protected against [11]:

- An authorised user of the system gaining access to information that should be hidden from them.
- A user masquerading as someone else, and so obtaining access to whatever that user is authorised to do. Any actions that they carry out are also attributed to the wrong person.
- Security controls being bypassed.
- Eavesdropping on a communication line, thus gaining access to confidential data.
- Tampering with the communication between objects - modifying, inserting and deleting transmitted data.
- Lack of accountability due, for example, to inadequate identification of users.

These threats are prevented through the securing of all communication over the network, the correct authentication of both client and server, and the authorisation of clients to access resources.

4.1 Secure Communication

Secure communication is provided through both the encryption and the digital signing of the data being passed over the network.

Encryption

Encryption prevents eavesdropping of sensitive data transmitted over an insecure network. There are two types of encryption algorithms - *asymmetric* and *symmetric* [17]. Asymmetric, or public-key, algorithms have two keys, a public and a private. Data is encrypted with one key and decrypted with the other. Symmetric encryption, or private key, however, uses just one key, agreed between the sender and recipient beforehand. This method is faster than asymmetric encryption. A common cryptographic technique is to encrypt each individual conversation with a separate key [17], called a *session key*. Session keys are useful because they eliminate the need to store keys between sessions, reducing the likelihood that the key might be compromised. However the main problem with session keys is their exchange between the two conversants. This is solved through the use of either a public-key cipher or key agreement algorithm, such as Diffie-Hellman.

Digital Signatures

Digitally signing data prevents the unauthorised modification of data during transmission. Rather than sign the whole message, which would add too much overhead, a unique *digest* is signed instead. This message digest is produced by a secure hashing algorithm, which outputs a fixed-length hash, unique to that message. This ensures that if the message is modified in any way, the digest will change, which can then be detected at the receiving end.

Digitally signing data is accomplished through the use of a public-key cipher. The message digest is encrypted with the private key of the sender, which the recipient deciphers using the senders public-key. If the data was modified, or the digest was encrypted with a different private key, then the original digest and the digest calculated by the recipient will be different. This ensures that both the data was not modified during transmission and that it originates from the true sender.

4.2 Authentication

In distributed systems, authentication is the means by which the identities of servers and clients are reliably established [8]. Authentication prevents unauthorised users from gaining access to the system and also prevents authorised users masquerading as another user. The mechanism used to achieve this is based on the possession of encryption keys. The fact that a principal possesses a private key is proof of the identity of that principal. Coulouris *et al.* [4] states that Needham and Schroeder first suggested a solution to authentication based on an *authentication server*. The authentication server maintains a list of private keys for all clients and servers. When a client wishes to communicate securely with a server, the authentication server generates a private session key and sends two copies of it to the client, one encrypted in the client's private key, and the other encrypted in the server's private key. The client then forwards on the server's copy. The identity of the client and server is authenticated, as only they know their private key needed to decrypt the session key. Digital certificates are also used as a means of authentication through the use of public-key cryptography [17]. The certificate includes the username and public key of the sender and is signed with the private key of a trusted authority known as a Certification Authority (CA). This allows the certificate to be authenticated with the CA's public key, which is widely available, without the need to contact the CA.

4.3 Authorisation

Authorisation is the granting of access to a particular resource [13]. This prevents both unauthorised and authorised users gaining access to information that should be protected from them. Authorisation is commonly implemented using Access Control Lists (ACL's), which is are a list of users associated with each file in the system, specifying who may access the file and how [22].

4.4 Kerberos

Kerberos is a distributed authentication service that allows a process (a client) running on behalf of a principal (a user) to prove its identity to a verifier (an application server) without sending data across the network that might allow an attacker or the verifier to subsequently impersonate the principal [10,17]. Kerberos optionally provides integrity and confidentiality for data sent between the client and server. Kerberos deals with three kinds of security objects:

- *Ticket*: a token issued to a client by the Kerberos ticket-granting service for presentation to a particular server, verifying that the sender has been recently authenticated by Kerberos. Tickets include an expiry time and a newly-generated session key for use by the client and the server.

- *Authenticator*: a token constructed by the client and sent to the server to prove the identity of the user and the currency of any communication with a server. An authenticator can be used only once. It contains the client's name and a timestamp and is encrypted in the appropriate session key.

- *Session Key*: a secret key randomly generated by Kerberos and issued to a client for use when communicating with a particular server. Encryption is not mandatory for all communication with servers; the session key is used for encrypting communication with those servers that demand it and for encrypting all authenticators.

Clients processes must possess a ticket and a session key for each server that they use. It would be impractical to supply a new ticket and key for each client-server interaction, so most tickets are granted to clients with a lifetime of several hours so that they can be used for interaction with a particular server until they expire. Timestamps also guards against the replay of old messages intercepted in the network or the reuse of old tickets found lying in the memory of machines from which the authorised user has logged-out.

4.5 Secure Sockets Layer

The SSL (Secure Sockets Layer) Handshake Protocol [9] was developed by Netscape Communications Corporation to provide security and privacy over the Internet. The protocol supports server and client authentication. The SSL protocol is application independent, allowing protocols like HTTP, FTP, and Telnet to be layered on top of it transparently. The SSL protocol is able to negotiate encryption keys as well as authenticate the server before data is exchanged by the higher-level application. The SSL protocol maintains the security and integrity of the transmission channel by using encryption, authentication and message authentication codes.

The SSL Handshake Protocol consists of two phases, server authentication and client authentication, with the second phase being optional. In the first phase, the server, in response to a client's request, sends its certificate and its cipher preferences. The client then generates a master key, which it encrypts with the server's public key, and transmits the encrypted master key to the server. The server recovers the master key and authenticates itself to the client by returning a message encrypted with the master key. Subsequent data is encrypted with keys derived from this master key. In the optional second phase, the server sends a challenge to the client. The client authenticates itself to the server by returning the client's digital signature on the challenge, as well as its public-key certificate.

5. The Java File System

This section outlines the design of the Java File System (JFS), which meets the requirements set out in section 2. The architecture for JFS is illustrated in figure 1.

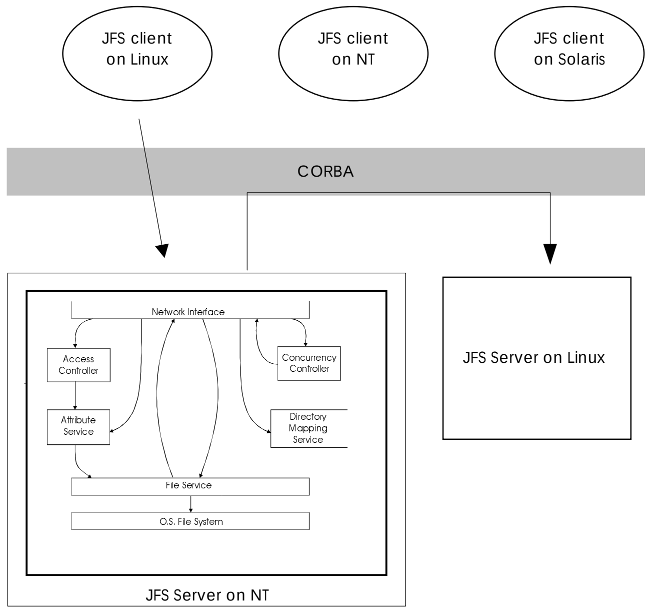

Figure 1 – JFS Architecture

5.1 Java File Server

Although the client can only ever connect to one server directly, the file service may in fact be distributed across a number of servers. The server that the client is connected to may have a number of its directories *mapped* to directories on remote servers. This distribution is transparent to the client however, who just sees one global tree structure.

The File Server consists of six main components, namely; Network Interface, Access Controller, Concurrency Controller, Attribute Service, Directory Mapping Service, File Service.

The **Network Interface** component implements the server side of the communications layer. It is responsible for the marshalling of data prior to transmission on the network and the unmarshalling of data received from remote hosts. In this way it provides an abstraction of the communication protocol from all the other components, thus enabling the protocol to be easily changed if required.

The **Access Controller** regulates a clients' access to the files stored in the Java File System. This is achieved through the use of Access Control Lists (ACL's). When a client requests to perform an operation on a file, The file's ACL is obtained from the Attribute Service, and the Access Controller can then verify that the client has sufficient permissions to complete the operation requested on the file.

The purpose of the **Attribute Service** is to maintain a set of attributes associated with each file. The Java language provides access to a number of basic attributes which are controlled by the operating system, such as the size of a file and the date it was last modified. However a number of extended attributes such as the owner of a file, and access control lists (ACL's), which are also required by the Java File System, are not supported by Java. This is due to Java's dependency on attributes provided by the host operating system.

The **File Service** provides the facility for all the other components of the Java File Server to access and modify files and directories in the Java File System. It is the only component in the server that can directly access the files stored on the server. All other components must access files through the File Service. The File Service also adds a number of extra file handling features, not implemented by Java. These include the ability of creating symbolic link files (a type of file which contains no data, but points to another file). These can be used to create links across multiple servers, running different operating system types.

The function of the **Directory Mapping Service** is to resolve a file's pathname from the Java File System tree structure into the server where the file resides and the location on that server. It does this through the use of a *mapping table,* similar to NFS, which is loaded into memory on initialisation of the server. This table contains a list of directories that have been mapped, along with the server where they map to and the location on that server.

5.2 Security Service

As discussed in Section 4, the main mechanisms which need to be implemented in order to secure a distributed system are *authentication*, *authorisation* and *secure communication*. Authorisation is already implemented in the Access Controller component of the Java File Server, therefore only authentication and secure communication need to be addressed

Authentication

Before a client is allowed to access any of the data stored in the file server, it must be able to prove its identity to the server. A secure *mutual identification* scheme, which does not require the transmission of passwords over the network, and provides authentication of both client and server was designed. This scheme is similar to the model provided by Kerberos, and is described below.

Coulouris *et al.* [4] state that a secret key is the equivalent of the password used to authenticate users in centralised systems. Therefore, possession of an *authentication* key, based on the client's password can verify the identity of both the client and the server.

The method used to convert the password into an encryption key is based on the method used in the PKCS #5 [15] standard, whereby the key is generated by computing the hash of the password. The method used in the Java File System differs only in that SHA-1 [50], is used as the hash algorithm. A user identifier is used to identify the client for a particular session. This identifier can be encrypted with the authentication key to produce a *token*. This is illustrated in Figure 2.

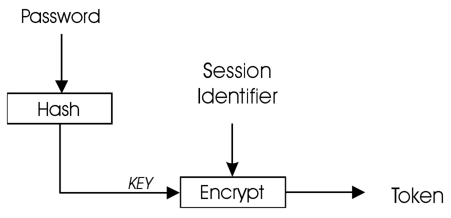

Figure 2 - Token Generation

Key Agreement

Before any data transmitted between client and server can be encrypted, a shared secret key between the two parties must be agreed upon. The authentication key generated to verify the identity of the client and server could be used as the secret key, however a random session key would be cryptographically more secure, as a longer key length could be used with a greater randomness than a digest. Also a different key would be used with each session. The Diffie-Hellman algorithm [14] is used for generating this shared secret key.

The Diffie-Hellman algorithm is however vulnerable to a *middleperson* attack. To overcome this, each message in the key agreement stage can be ‘signed’ using the authentication key generated above. To do this, each message in the key agreement sequence is encrypted in the authentication key, generated above. This allows the receiving party to verify the source of the message, and prevents an intruder from substituting their own messages.

Network Traffic Encryption

The security architecture is controlled by a *Security Manager* located in both the client and the server. When the client first connects to the server, the Security Manager is responsible for both authentication and session key agreement, as described above. Once these have been performed, it is also responsible for the encryption and decryption of all data transmitted over the network. RC4 was chosen as the encryption algorithm as it is one of the fastest symmetric ciphers available [17]. In addition to encrypting all requests, they are also signed to prevent unauthorised modification which being transmitted.

The steps required for a secure communications protocol are outlined below and illustrated in figure 3.

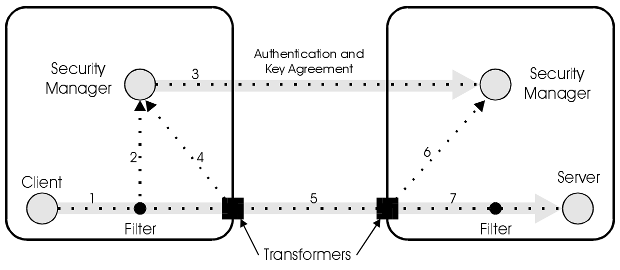

Figure 3 - Security Design

1. The client issues a request to the File Server

2. The filter intercepts the request and announces it to the Security Manager.

3. The Security Manager determines if a session already exists between the client and server. If it is the first call to the server, a new session needs to be negotiated with the server. The client’ s Security Manager contacts the server’s Security Manager, and performs the authentication and key agreement sequences, as described above.

4. The request is intercepted by the transformer just prior to being put on the network, and is passed to the Security Manager for encryption. The Security Manager locates the correct session key and uses it to encrypt the complete request. A header is then attached to the request containing the session id, the protocol used for the encryption and the message signature.

5. This extended request is then written to the network.

6. The request arrives at the server and is intercepted at the server's transformer. It is then passed to the server's Security Manager for decryption. The Security Manager uses the session identifier in the request header to locate the correct session key. It is then able to decrypt the request using this key.

7. The request arrives at the server object.

In addition to encryption, all requests and replies are signed by encrypting their digest with the authentication key. This allows the receiver to verify the source of the request, and ensure that the request was not modified during transmission.

5.3 Concurrency Control

There are two types of locks that a client can hold on a file, a *read lock* or a *write lock.* Read locks are shared, and allow multiple clients to read the same file concurrently. Write locks on the other hand are exclusive, and no other client is allowed to hold either a read or a write lock on that file concurrently. Clients are not allowed to block, waiting for a lock. If for example, a client wishes to write to a file that is currently being read by another client, then the server does not block the writer client until the reader client has finished. It simply returns an error to the client as the file cannot be written to. This prevents deadlock.

5.4 Client Applet

The applet consists of two parts, namely the network interface which communicates directly with the server and the graphical file manager front-end. The applet can only be used to access files stored on the remote server, and be cannot used to access files on the local host.

5.5 Implementation Issues

We have successfully implemented a fully working version of JFS using JDK 1.2 and the OrbixWeb 2.0.1 CORBA implementation [12]. Figure 3 illustrates the Java File Manager applet, which can be used to manipulate the heterogeneous distributed file system, comprised of a number of different file systems, running on different operating systems. A small set of classes, downloaded with the client application, is all that is required to access the remote file server.

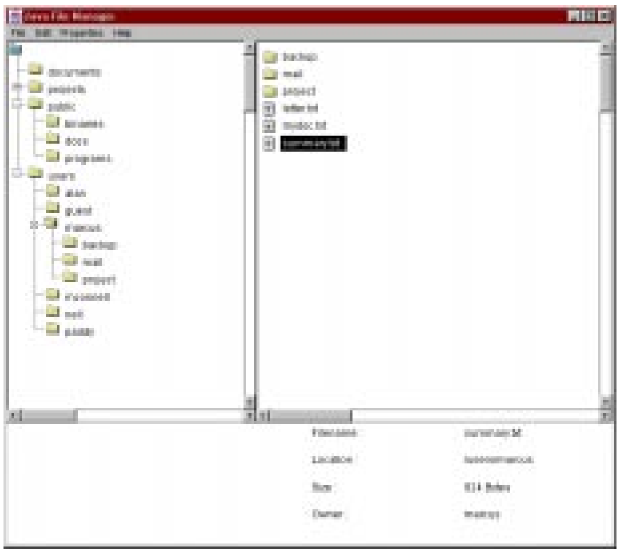

Figure 4 - Java File Manager applet

The security transformers were implemented using OrbixWeb Transformer objects, which allow access to CORBA request data immediately prior to transmission over the network. This ensures that the entire request is encrypted and not just the request data. Had transformers not been used, parts of the request, such as the method call, would have remained unencrypted, thus reducing security.

A possible extension to the JFS architecture which would improve performance, would be the caching of data on the client. Although the client cannot implement a large secondary cache, such as that implemented in the Andrew File System, a memory-based cache could significantly reduce the number of network calls required. The contents of a directory, for example, changes infrequently, but the client nevertheless must still obtain a fresh view from the server every time the directory is opened. The introduction of caching could however lead to significant problems in other areas. The storage of sensitive data in memory could introduce a serious security weakness. If the Java Virtual Machine pages this memory to disk for example, the sensitive data stored in the cache would then be vulnerable to attack. Some form of memory obfuscation would therefore be required. Another area where the introduction of caching could lead to problems is the potential for clients to see *out-of-date* data. Therefore, a comprehensive cache coherence protocol, such as that provided by the Andrew File System, would be required.

6. Conclusion

This paper presents the Java File System, a heterogeneous distributed file system designed to meet the goals outlined in section 2. JFS provides a distributed file system to clients, which is also heterogeneous in both hardware and operating system type. JFS also provides a security architecture in which both client and server are authenticated using a mutual authentication scheme, access to files is regulated through Access Control Lists, and all communication over the network is encrypted though the use of both public-key and private-key cryptography. In addition, JFS also manages concurrent access to files, providing both exclusive and shared locks on files.

There is an initial performance hit when a client first connects to the server, due to exponentiation in the Diffie-Hellman algorithm. However, once the initial key agreement is completed, there is negligible performance loss due to encryption. This is due mainly to the use of the RC4 algorithm, which is one of the fastest symmetric ciphers available [17]. This performance hit is however a small sacrifice, given the benefits provided by the security system.

JFS demonstrates the viability of a Java-based file system and also fills a niche where use of the NFS would not be feasible. JFS also provides a strong security model, which provides more robust authentication and encryption that that supported by both NFS and WebNFS.

7. References

[1] Brent Callaghan. October 1996. *WebNFS Server Specification.* RFC 2055. ftp://ftp.isi.edu/in-notes/rfc2055.txt

[2] Brent Callaghan. October 1996. *WebNFS Client Specification.* RFC 2054. ftp://ftp.isi.edu/in-notes/rfc2054.txt

[3] Cattaneo and G. Persiano. *Design and Implementation of a Transparent Cryptographic File System for Unix.* Università di Salerno, Baronissi, Italy. http://edu-gw.dia.unisa.it/tcfs/

[4] George Coulouris, Jean Dollimore, Tim Kindberg. 1994. *Distributed Systems, Concepts and Design, Second Edition.* Addison-Wesley.

[5] Federal Information Processing Standards. April 17 1995. *Secure Hash Standard.* FIPS PUB 180-1. http://www.itl.nist.gov/div897/pubs/fip180-1.htm.

[6] IONA Technologies PLC. 1997. *OrbixSecurity White Paper version 1.0.* http://www.iona.com/support/whitepapers/orbixsecurity/index.html

[7] IONA Technologies PLC. November 1996. *OrbixWeb version 2 Reference Guide.*

[8] Sape Mullender. 1993. *Distributed Systems, Second Edition.* Addison-Wesley.

[9] Netscape Communications Corporation. 1996. *The SSL Protocol version 3.0*. http://www.netscape.com/

[10] B. Clifford Neuman and Theodore Ts' O. September 1994. *Kerberos: An Authentication Service for Computer Networks.* IEEE Communications. Volume 32, No. 9

[11] Object Management Group.
November 1996. *CORBAservices Specification.*
http://www.omg.org/

[12] Marcus O'Connell. April 1998. *A Secure Distributed File System for Network Computers.* Trinity College Dublin.

[13] Open Software Foundation. 1991. *Security in a Distributed Computing Environment.* http://www.santix.de/dce/security.htm

[14] RSA Laboratories. November 1 1993. *PKCS #3 : Diffie-Hellman Key-Agreement Standard*. Version 1.4. ftp://ftp.rsa.com/pub/pkcs/ascii/pkcs-3.asc

[15] RSA Laboratories. November 1 1993. *PKCS #5 : Password-Based Encryption Standard*. Version 1.5. ftp://ftp.rsa.com/pub/pkcs/ascii/pkcs-5.asc

[16] Russel Sandberg. *The Sun Network File System: Design, Implementation and Experience.* Sun Microsystems, Inc.

[17] Bruce Schneier. 1996. *Applied Cryptography, Second Edition.* John Wiley & Sons.

[18] Sun Microsystems. 1997. *JavaStation white paper.* http://www.sun.com/javastation/whitepapers/javastation/javast_ch1.html

[19] Sun Microsystems. March 1989. *NFS : Network File System Protocol Specification.* RFC 1094. ftp://ftp.isi.edu/in-notes/rfc1094.txt

[20] Sun Microsystems. March 1995. *The NFS Distributed File Service, White Paper.* http://www.sun.com/solaris/wp-nfs.html

[21] Sun Microsystems. May 1996. *WebNFS - The Filesystem for the World Wide Web, White Paper.*

[22] Andrew Tannenbaum. 1995. *Distributed Operating Systems.* Prentice Hall, international edition.

[23] Edward R. Zayas. September 1991. *AFS-3 Programmer's Reference : Architectural Overeview. Version 1.0.* Transarc Corporation.