A Wide-Area Distribution Network for Free Software

Arno Bakker (contact)
Maarten van Steen
Andrew S. Tanenbaum

Internal report IR-CS-002, January 2003

Vrije Universiteit Amsterdam
Faculty of Sciences
Department of Computer Science
De Boelelaan 1081a
1081 HV Amsterdam
The Netherlands
Tel: +31-20-4447762
FAX: +31-20-4447653
E-mail: arno@cs.vu.nl

A Wide-Area Distribution Network for Free Software

ARNO BAKKER,
MAARTEN VAN STEEN,
and
ANDREW S. TANENBAUM
Vrije Universiteit, Amsterdam

January 30, 2003

Abstract

The Globe Distribution Network (GDN) is an application for the efficient, worldwide distribution of freely redistributable software packages. Distribution is made efficient by encapsulating the software into special distributed objects which efficiently replicate themselves near to the downloading clients. The Globe Distribution Network takes a novel, optimistic approach to stop the illegal distribution of copyrighted and illicit material via the network. Instead of having moderators check the packages at upload time, illegal content is removed and its uploader's access to the network permanently revoked only when the violation is discovered. Other protective measures defend the GDN against internal and external attacks at its availability. By exploiting the replication of the software and using fault-tolerant server software, the Globe Distribution Network achieves high availability. This article describes the design of the GDN and a number of small experiments with its prototype implementation.

1 Introduction

The scale of distributed applications can classified along three dimensions Neuman (1994). Numerically large applications have many users or many components. Geographically large applications have their users or components distributed over a large geographical area (e.g. worldwide); and an administratively large application entails that many organizations are involved in the application as users or as administrators of its components.

Developing an Internet application that is large in any of the three dimensions is difficult. Dealing with millions of users and components introduces many engineering and management issues. It requires the extensive use of techniques such as caching, replication, and distribution of functionality to reduce and distribute the load over the available infrastructure Neuman (1994). These techniques, in turn, introduce technical and managerial problems of their own, such as maintaining consistency of caches and replicas, and how to keep track of a (replicated) component's current location(s). Large geographical distances introduce unavoidable and significant communication delays, whose impact again have to be minimized by caching, replication, and distribution of functionality. Having to deal with many organizations makes it hard to administer and secure the application, in particular, if these organizations operate in different parts of the world. In addition to the problems introduced by the large scale of the applications, a developer also has to deal with machine and network failures, and heterogeneity in hardware and (system) software.

The key to making large-scale application development easier is therefore to provide the developer with the means for dealing with these complex (nonfunctional) aspects and required techniques in a comprehensive manner. Particularly important for a development platform, in addition to comprehensiveness, is flexibility. To build an application with hundreds of millions of users operating on a worldwide scale, it is necessary that the development platform allows the developer to employ the techniques, protocols and policies that are best suited for the application Agha (2002). This implies that the platform should support many different mechanisms and policies and it should also allow new ones to be introduced easily. In short, to accommodate large-scale applications, a platform should allow application-specific optimizations of the middleware itself.

The Globe project is aimed at designing and building such a comprehensive and flexible middleware platform Van Steen et al. (1999). Flexibility is achieved by basing the middleware platform on a new model of distributed objects, called the *distributed shared object model*. A distributed shared object is in control of all aspects of its implementation, including nonfunctional aspects such as replication protocol and security. A distributed shared object can therefore be said to bring its own middleware to the machines it uses, and thus enables a developer to also apply application- or even object-specific solutions in the nonfunctional aspects of the object's implementation.

To validate its design, we have built several applications on top of the Globe middleware. One such application is the *Globe Distribution Network (GDN)*, the design of which is the topic of this article. The Globe Distribution Network is an application for the efficient, worldwide distribution of freely redistributable software packages, such as the GNU C compiler, the Apache Web server, Linux distributions and a great deal of shareware Bakker et al. (2000); Bakker (2002). The distribution is made efficient by encapsulating the free software in Globe distributed shared objects that automatically replicate themselves to areas with many downloaders. The use of distributed shared objects to replicate software not only makes its distribution efficient, it also allows us to remedy some of the problems of FTP- and HTTP-based software distribution. In particular, distributed shared objects allow transparent fail-over to other replicas and can guarantee strong replica consistency, unlike many pull-based mirroring solutions.

In addition to efficiency and fault tolerance, the design of the Globe Distribution Network pays particular attention to the security aspects of software distribution. One of the most pressing legal problems concerning the Internet today is the illegal distribution of copyrighted works, such as digitized audio and video, and illicit content, such as child pornography. As the GDN is aimed at legitimate large-scale file sharing and is intended to be open to many free-software publishers, it takes measures to prevent illegal distribution. Our approach to inhibiting this type of abuse is novel and necessarily optimistic (in the sense of optimistic concurrency control). Rather than preventing illegal distribution of content by moderation beforehand, we make content traceable to its publisher, and remove content and block its publisher only afterwards, when the publication has proven illegal. This optimistic approach is necessary to keep the amount of work for the volunteers operating the nonprofit Globe Distribution Network low. Other security aspects that are taken into account in the design of the GDN are authenticity and integrity of the software being distributed, and denial-of-service attacks by external and internal attackers.

The remainder of this article is structured as follows. Section 2 describes the architecture of the Globe Distribution Network, and how it uses distributed shared objects to make software distribution efficient. Section 3 presents our approach to preventing illegal distribution. In Section 4 we describe how the authenticity and integrity of the free software being distributed is ensured. Section 5 describes how we guarantee the availability of the GDN despite attacks by insiders and outsiders. The fault tolerance measures of the GDN are presented in Section 6, and Section 7 briefly looks at the performance of the GDN. We conclude in Section 8.

2 Architecture

The architecture of the Globe Distribution Network is shown in Figure 1. The GDN consists of a large number of distributed shared objects (DSOs) encapsulating the software being distributed. The distributed shared objects are replicated over a collection of *object servers* located throughout the Internet. An object server is a

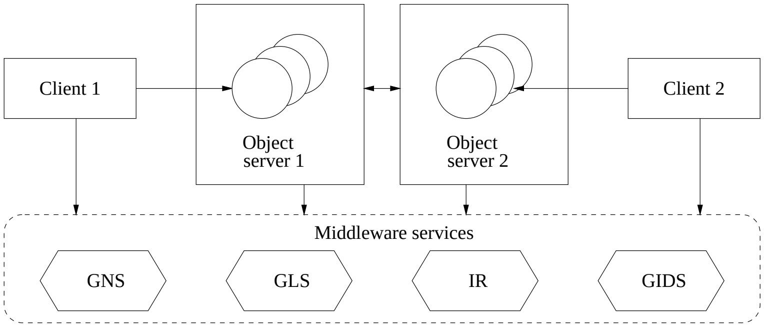

Figure 1: The architecture of the Globe Distribution Network. A circle represents a replica of a distributed shared object; a rectangle represents a machine; a diamond represents a service, implemented by one or more machines; a dashed rounded box represents a collection.

user-level server process that is capable of hosting replicas of distributed shared objects. The clients, distributed shared objects and object servers are supported by a number of middleware services.

2.1 Downloading Software

In this architecture a download proceeds as follows. To download software from a distributed shared object, the client first asks the *Globe Name Service (GNS)* to resolve the symbolic name of the object to the object's *object handle*, which is the object's permanent, location-independent (but binary) identifier. Second, the client resolves the object handle of the object to the *contact address* of the nearest convenient replica of the object. The contact address contains the network address of the replica and information on how to contact it (i.e., which protocols to use in communication). The object handle is resolved to a contact address using the *Globe Location Service (GLS)*. The GLS returns the contact address of the geographically or network-topologically nearest replica, depending on the setup of the GLS. It has been specifically designed to track replicas for billions of objects, and has lookup costs that are proportional to the distance between lookup requester and nearest replica Van Steen et al. (1998); Ballintijn et al. (2001).

In the third step, the contact address of the replica is used to construct a proxy of the object in the client's address space. The code for the proxy and the protocols used is dynamically loaded from a trusted *implementation repository (IR)*. Finally, the client downloads the software from the object by invoking methods on the proxy which are then shipped to and executed at the nearby replica. This replica reads the software from local storage and returns it to the client.

Software uploads proceed in a similar manner. An uploading client installs a proxy in its address space, and subsequently invokes the object's `startFileAdditon`, `putFileContent` and `endFileAddition` methods to upload the file containing the free software into the object. These state-modifying methods are executed by all replicas of the object, which write the file to local storage.

2.2 The Structure of Distributed Shared Objects

The proxies and replicas of a distributed shared object are composed of *subobjects*, modules that take care of a particular aspect of the object's implementation. A replica of an object minimally consists of four subobjects, as illustrated in Figure 2. The *replication subobject* (labeled R in the figure) contains the implementation of the replication protocol used by this object. The *communication subobject* (Co in the figure) satisfies the replication subobject's communication needs, for example, by offering reliable group communication primitives. The *semantics subobject*, labeled S in the figure, contains the actual implementation of the object's methods and (logically) holds the state of the object. Finally, the *control subobject* (labeled Ct) manages the interaction between the replication protocol and the object implementation; it bridges the gap between the application-defined interfaces of the semantics subobject and the standardized interface of the replication subobject. More sophisticated proxies and replicas also contain subobjects for handling security and fault-tolerance aspects.

The modular structure enables the application developer to change the subobjects that handle replication or other nonfunctional aspects on a per-object basis. In other words, it allows the developer to select different protocols, techniques and policies for different objects, thus achieving our middleware's goal of being able to select the solutions best-suited for each application. The standardized interfaces of these subobjects furthermore allow us to build a large library of subobjects that are reusable by other applications.

2.3 Efficient Distribution via DSOs

In the Globe Distribution Network, the distributed shared objects use an intelligent replication protocol that ensures efficient distribution of the free software. We define efficient distribution as distribution that avoids frequent communication over wide-area network links, where bandwidth is assumed to be scarce. To avoid wide-area links, the replication protocol monitors access patterns, and replicates

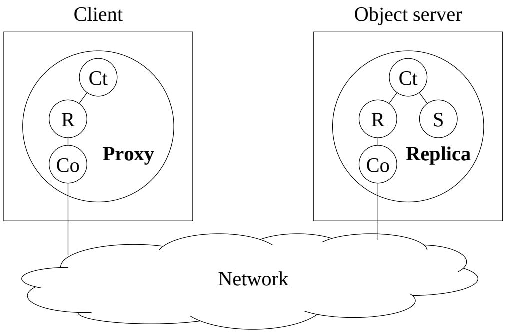

Figure 2: Composition of proxies and replicas. A large circle represents a proxy or replica of a distributed shared object; a small circle represents a subobject; a rectangle represents a machine.

the object in areas with many downloads, thus bringing the software near to the clients.

More specifically, when a client invokes a method on the object the client reports its location to the nearby replica. The replica aggregates the location statistics and periodically determines which regions are generating many requests. If the number of requests per time interval exceeds a certain threshold, the replica autonomously creates a new replica in that region. Replicas not only evaluate their load periodically, they also detect *flash crowds*, that is, sudden sharp increases in the number of accesses which are common on the Internet Nielsen (1995), and create extra replicas in response. To find available object servers to create replicas on, the replicas query the *Globe Infrastructure Directory Service (GIDS)*. This middleware service is used to register the object servers participating in the Globe Distribution Network, and can locate an object server willing to host object replicas in a particular region of the Internet Kuz et al. (2002). Each distributed shared object starts out with a few trusted *core replicas*, which coordinate global operations (such as state updates) in the object.

3 Illegal Distribution

Before describing how the Globe Distribution Network deals with illegal distribution, we first define more precisely what constitutes illegal distribution in the context of free software. Software is generally considered a literary work and hence protected by copyright World Intellectual Property Organization (1996).[1] Allowing free redistribution of software therefore requires the legal consent of the copyright holder, generally the author. For software, standard licenses have emerged under which authors can publish their software and which, amongst other things, permit free distribution. A well-known license is the GNU General Public License (GPL) Free Software Foundation, Inc. (1991)

The above suggests that legal distribution of free software can be defined as the distribution of software that has been made freely redistributable by its copyright holder. This definition is, however, too broad, as not all types of software can be legally owned and distributed. We define *controversial free software* as software that

1. Uses patented technology,
2. Can be used to circumvent copyright-protection measures,
3. Employs strong cryptography, or
4. Contains racist or potentially offensive material (e.g. Nazi symbols, nudity).

Whether or not instances of this class of software can be legally owned and distributed differs from country to country. For software with patented technology it depends on the geographical scope of the software patents violated whether or not the software is illicit in a country. The legality of the second type depends on the current copyright laws of the country in question. Software whose sole purpose is the circumvention of copyright protection is illegal in most countries, but for software that has legitimate purposes as well ("dual use software"), different countries have different regulations. A famous example in this category is the DeCSS code, which can be used both for playing Digital Versatile Discs (DVDs) on open-source operating systems and for obtaining the unprotected video material from the disc, thus allowing lossless and unrestricted further copying Simons (2000).

In many countries the use and ownership of software that uses strong cryptography is legal, and distribution is allowed except to certain nation states. Furthermore, the distribution of strong cryptographic software generally does not require the original publisher to follow special procedures. A notable exception is the United States U.S. Department of Commerce (2001). Finally, for software containing controversial material, it also depends on local legislation whether this software can be owned and distributed.

The Globe Distribution Network does not, at present, address per-country differences with respect to which software is illegal. These issues require further

[1] For an interesting discussion about whether software should be protected under copyright law, patent law, or free speech legislation, see Burk (2001).

investigation. In the meantime, we define our own global policy of what can be distributed via the GDN. Given that the GDN is to be used for the distribution of free software, we consider inappropriate content as anything that is not freely redistributable software or part thereof.

3.1 Prevention Schemes

The Globe Distribution Network is intended to support a large number of software publishers. This design goal has important consequences, as it implies that a powerful distribution channel is made available to many people, some (or many) of whom will attempt to abuse this channel to illegally distribute copyrighted or illicit content. The actions of these abusers can create liabilities for the operators of the distribution network, in particular, for the owners of the object servers storing and serving the GDN's contents. For example, in the United States the owner of a computer himself is liable for copyright infringement when copyrighted content is served from his computer, rather than the person that put it there United States Government (1998). A similar risk exists for pornography and other illicit materials.

These liability risks make it particularly important for the GDN to have adequate measures against illegal distribution, as the GDN is intended to be operated by volunteers. As indicated in the introduction we want the server and network resources to be donated and the management of the application to be performed by volunteers, as in the current FTP-based infrastructure. The problem is that if there is a risk of liability, people and organizations may not volunteer their time and resources towards the GDN. As a result, it may not be possible to find sufficient people and resources to setup a useful worldwide distribution network.

3.1.1 Content Moderation

The most obvious solution to preventing illegal distribution is to check content before it is uploaded onto the distribution network. We call this solution *content moderation*. In content moderation, one or more people, called *moderators*, manually check all content before it is allowed onto the distribution network. Manual checking is required because a computer cannot tell copyrighted from noncopyrighted content nor illicit from legal content.

We deemed content moderation inappropriate as a preventive measure for the Globe Distribution Network for two reasons. First, finding a group of moderators that are willing to devote their time may be hard. The manual checks are tedious and time consuming, in particular, the checks that require manual inspection of the source code, such as checking whether or not a software package is controversial software. What makes being a content moderator most unattractive is that, when abuse is low, most of the moderation work is done in vain. As a result, we expect moderators to perceive the moderation work as superfluous, become demotivated, and no longer volunteer their time for moderating the network's content. Also a moderator who accidentally approves illicit content may have some legal liability for this mistake.

Second, content moderation introduces a (potentially long) delay between the initial submission for publication and the actual publication in the distribution network, especially if the moderator has to check many patents, court rulings, etc. We expect that software maintainers wanting to publish via the GDN will find this delay irritating.

3.1.2 Reputation

The reputation of a software publisher can also be used as a method for preventing illegal distribution. Currently, people who want to setup a mirror of free software often select a collection of software packages or distributions based on the good reputation of the author/publisher and configure their servers to directly mirror the primary publication sites. This method is applied, for example, for the Linux kernel and for well-known distributions such as RedHat and FreeBSD. The disadvantage of the current practice is, however, that each site owner has to monitor the reputation of software publishers to see who deserves to be mirrored. This method can be improved upon by introducing a group of reviewers who maintain a list of trusted publishers. Assuming some form of dynamic replication as in the GDN, site owners would then configure their servers to automatically accept content but only from the publishers on the list.

With this method, gaining access to the network with the intention of abusing it is nearly impossible. To have software published by other sites, a malicious publisher not only has to establish a good reputation, but he has to also create a large user base such that the reviewers start considering the software as a candidate for replication. As establishing a reputation and creating an audience is time consuming and hard, the risk of abuse and thus illegal distribution in this method is low.

The reason we did not adopt this simple approach for the GDN is that it does not give equal access to all publishers. We would like each publisher to be able to immediately benefit from the GDN's resources and facilities when demand for his software grows. This reputation scheme can, however, be selectively employed in the GDN, as explained in Section 3.3.

3.1.3 Cease-and-desist

Our solution to preventing illegal distribution in the Globe Distribution Network is called *cease-and-desist* Bakker et al. (2001). In the cease-and-desist scheme, users of the distribution network can freely publish content, but the content a user publishes remains traceable to that user. If content is suspected of having been published illegally, its presence in the distribution network is reported to a group of moderators. This group of moderators checks whether or not the content was published illegally, and if so, blocks the publishing user's access to the distribution network and has his publications removed. In short, the cease-and-desist scheme tries to limit illegal distribution by banning provably malicious users from the distribution network. Intuitively, a distribution network using cease-and-desist becomes similar to a world-writable directory on a UNIX operating system: everybody can place files in the directory but the files always remain traceable to the user that put them there because of the associated ownership information. Users that put illegal content in the directory lose their account.

The cease-and-desist scheme works under three conditions.

1. A small amount of illegal distribution must be tolerated.

2. Persons banned from the distribution network must not be able to easily regain access.

3. The number of reports of illegal content to the moderators must be low when abuse is low.

We discuss these three conditions in turn. As illegal content will be removed from the distribution network only after it has been detected, there will always be a certain amount of illegal content available via the network. We argue that the law will have to accept this situation and allow a certain level of abuse, as there is no possibility to keep out all illegal content, in any scheme. Even content moderation, which the best scheme for limiting the amount of illegal content, cannot filter out all illegal content. The reason is that the detecting illegal content requires manual checks which are error prone and, furthermore, may be defeated by cleverly encoding illicit content into inconspicuous content. A powerful example supporting this latter argument is the so-called "first known illegal prime number", encoding the DeCSS source code Carmody (2002).

The second condition is that it should be impossible or at least difficult for a violator to regain access to the distribution network after he has been blocked. The following method was chosen to satisfy this requirement. Candidate users are required to prove their real-world identity, which is published on a black list when the user is found guilty of illegal distribution. This black list is checked at each application for access to the distribution network, thus keeping violators reapplying for access out.

The advantage of the cease-and-desist scheme over content moderation is that the amount of work for moderators can be small when the level of abuse is low. In that situation the work of the moderators is always useful and will not be perceived as superfluous. Cease-and-desist is therefore an optimistic scheme (in the sense of optimistic concurrency control) that is more labor-efficient at low abuse levels, whereas content moderation is pessimistic, and is thus also more effective when abuse is high. The GDN is sufficiently flexible to switch to content moderation when abuse is high, as discussed in Section 3.3.

However, to achieve this labor-efficient situation when abuse is low, there should be few false reports, which is the third condition specified above. The cease-and-desist scheme does not prescribe who should report illegal distribution, as this depends on legislation. In some cases, making moderators aware of copyrighted works in the distribution network is the legal responsibility of the copyright holders. Likewise, it could be the legal responsibility of law enforcement agencies to report illicit content. In other cases it may be the joint responsibility of the distribution network's users.

In the latter class of cases, false reports could be submitted by malicious people trying to undermine the prevention scheme. Their number can be kept at a minimum only if there is a threshold for a user for submitting a report. The most effective threshold is one where the user stands to lose something when the allegation in the report is false. The threshold we chose is to require that allegations are made by a user who will be blocked from the distribution network himself if the allegation proves false (i.e., he will lose his right to publish on the GDN or make allegations).

The cease-and-desist scheme for handling the problem of illegal distribution of copyrighted or illicit content is in line with current legal developments. For example, in the United States legislators have recognized that it is often not feasible to moderate content beforehand. Hence, in the recent changes to copyright law "provider[s] of online services," such as Internet Service Providers can request legal protection from copyright infringements by their users United States Government (1998). If a user publishes other people's copyrighted works on the ISP's servers, the ISP cannot be held liable and is required only to remove the copyrighted content once they have been notified by the copyright holders. France has similar legislation Oram (2001).

In the next section we explain how cease-and-desist is implemented in the Globe Distribution Network.

3.2 Cease-and-desist in the GDN

The cornerstone of the cease-and-desist scheme is that the published software remains traceable to its uploader, which we refer to as the *producer* of the software. Content traceability is implemented in the GDN as follows. When a producer wants to start publishing his software through the GDN he has to contact one of the so-called *access-granting organizations*. An access-granting organization, or *AGO* for short, verifies the candidate's identity by checking his passport or other formal means of identification. In addition, the organization checks if this person has not been banned from the GDN by any of the other AGOs by consulting a central black list.

If the candidate is clean, the access-granting organization creates a certificate linking the identity of the candidate to a candidate-supplied public key and digitally signs this certificate. This certificate is called the *trace certificate* and the key pair of which the public key on this certificate is one part is called the *trace key pair*. In addition to creating a trace certificate, the AGO supplies the producer with Globe security credentials that allow him to access the GDN.

An owner of an object server specifies which producers it wants to give access to his object server. In principle, access is granted at AGO-granularity: the server owner specifies which AGOs it trusts to do a proper identity and black-list check, and only producers that have credentials and certificates signed by those AGOs will be allowed to place content on that owner's object server. Owners can also give access to or block individual producers.

3.2.1 Uploading Content

An upload now proceeds as follows. Assume the producer has created a distributed shared object on one of the object servers that trusts the AGO the producer got his credentials from. Before uploading a file into the GDN, the producer creates a digital signature for this file using the trace key pair. This signature is referred to as the *trace signature*. The trace signature and associated trace certificate are uploaded into the DSO along with the file, by invoking the DSO's upload methods.

When the upload is finished, the DSO verifies the trace signature. If the signature is false, either because the certificate did not contain the right public key, the file did not match the digital signature, or the producer has been banned from the GDN, the object removes the uploaded file from its state. As only those files are allowed that are provided by an active (i.e., nonblocked) producer and that carry a valid signature, all content in the GDN is always traceable.

The whole procedure is summarized in Figure 3. To get access to the GDN a software producer first identifies himself to an AGO and receives a trace certificate

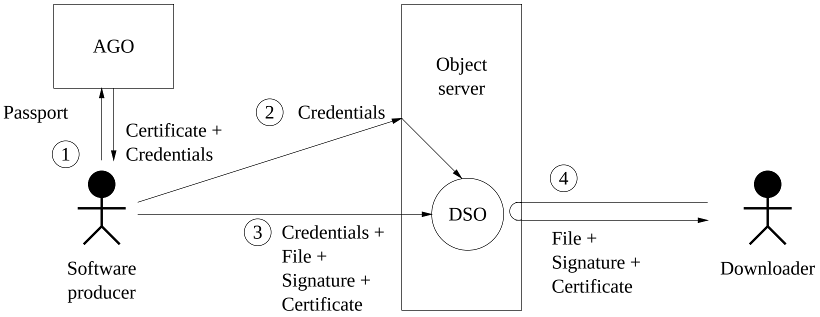

Figure 3: Basic operation of the GDN with traceable content.

and Globe credentials in return (arrow 1 in Figure 3). Second, the producer requests an object server, using his Globe credentials, to create a DSO (arrow 2). Third, the producer creates a digital signature for the file to be published and uploads it along with the file and the trace certificate into the DSO (arrow 3). Finally, a user downloads the file, trace certificate and trace signature from the DSO and subsequently verifies that they match (arrow 4) as we explain in Section 4.

3.2.2 Blocking Illegal Publishers

To ban a producer from the GDN when someone finds illicit content that is traceable to him, the following procedure is executed. When a regular user finds illicit content in the GDN he contacts a GDN producer who will make the accusation on his behalf. The accusing producer notifies all object-server owners and the access-granting organization that gave the suspected producer access of the publication of illicit content by the suspect. The access-granting organization, in addition, receives a copy of the signed illicit content, and verifies that this content is indeed inappropriate and is digitally signed by the violator. If this is the case, the producer's credentials are revoked and the violator is placed on the central black list shared by all AGOs. In effect the violator is thus banned from the GDN. On the other hand, the accusing producer is banned himself if the accusation he makes proves false.

The actions taken by the object-server owners upon notification depend on their *content-removal policy*. They may destroy their replicas of all objects that contain content signed by the violator, or delete the replicas of only the objects mentioned in the allegation. They may do so immediately upon notification by the accusing producer, or only after the allegation has been verified by the AGO. Object-server owners can also decide not to remove the content but instead temporarily block accused producers from their server.

Which policy object-server owners will adopt depends on the requirements imposed by the law, the level of abuse in the GDN and whether or not people report the abuse. In principle, object-server owners are autonomous and can decide for themselves which policy they adopt. However, the GDN may also impose a global policy to guarantee certain systemwide properties with respect to illegal distribution. We currently require object servers to follow a global policy by which all content published by a violator is deleted, but only after verification of the evidence. This policy provides protection against malicious producers trying to remove well-known software packages from the GDN.

3.3 Discussion

In Section 3.1.3 we identified three conditions that should be met to successfully use the cease-and-desist scheme. The second condition is to make sure that people who have been previously banned from the distribution network cannot regain access. In the GDN this problem is solved by requiring candidate producers to formally prove their real-world identity. This identity is revealed on a (public) central black list shared by all access-granting organizations for the GDN if the producer is found guilty of violating GDN's policy with respect to content.

Formal means of identification are necessary as they cannot be easily faked. It is assumed that the candidate presents his passport or other identification means to a local representative of the access-granting organization, or sends a copy of the document to the AGO, certified by a mutually trusted third party. Public-key certificates from well-known certification authorities may also be acceptable if they were issued only after sufficient identity checks.

Other methods of access control are also possible, for example, requiring the endorsement of a certain number of other GDN producers, or a refundable fee which is lost when the producer violates the rules. Which methods for blocking access are permitted depends on the legal requirements, as some methods make it easier to regain access than others. For example, the law may require publishers of child pornography to be locked out permanently. Our present standpoint is a conservative one, where we try to limit the amount of illicit content in the network as much as possible, and we therefore choose to directly and permanently block violators. This standpoint also means that, at present, we do not allow for fair mistakes, for example, by blocking a violator only after n violations, as allowing for mistakes would increase the level of illegal content in the network.

To what level a candidate has to prove his identity can differ per access-granting organization. The policy specifying the AGO's requirements is called its *access-granting policy*. For example, a group of object-server owners could setup their own AGO and define their own access-granting policies and thus their own rules about who gets access to their servers. A minimum requirement for an access-granting policy for an AGO in the GDN is, however, that the AGO is able to reveal a producer's real name when that producer is proven to illegally distribute content.

The relationship between a producer and the AGOs that gave him access should be viewed as a contract. A producer is given access to the GDN in exchange for which he promises not to abuse this right. If he does, it is the other party in the contract, the AGO, that establishes this fact and terminates the contract on this ground. In principle, the contract requires a GDN producer not to violate the GDN's global policy on what content may or may not be allowed. An AGO could, however, also impose additional restrictions on content and revoke only the credential it gave the producer instead of blocking him at all AGOs.

The third condition for cease-and-desist to work is to keep the number of false reports to the moderators low when abuse is low. The moderator task is in this solution carried out by the AGOs. If reports are submitted by regular users rather than copyright holders or law enforcement, this is a nontrivial problem. In our solution, only active (i.e., nonblocked) GDN producers may send in allegations, implying that regular users who are not producers themselves need to make the accusation through another active GDN producer. The reason for imposing this requirement is to create a threshold for users that will keep them from submitting false reports: they or their intermediaries will lose their access if they do. It may, however, not be necessary to block a false accuser the first time round or block him permanently. One could allow for a few mistakes and revoke access just for a period of time, as long as the number of false reports made remains low.

It is in the interest of the accusing producer to make these accusations, as in the long run, not participating in banning malicious producers will result in the collapse of the GDN and deprive the accusing producer of a cheap distribution channel for his own software. In other words, making an accusation, possibly on behalf of other users, is the price producers have to pay for access to the resources of the GDN. One can also imagine people specializing in the role of accuser, that is, producers acting as public prosecutors and explicitly requesting users to report illicit content to them.

Alternative methods for keeping false allegations low are similar to those for keeping violators out of the GDN, for example, a refundable fee which is lost when the accusation is false. Requiring that a group of users or producers make the allegation may not be a good alternative to blocking the false accuser. The effectiveness of the group method depends on the number of malicious persons in the user community. If there are many and they are organized it will be easy for them to swamp the AGOs with false reports. This method basically lacks a way to stop malicious users or producers from repeatedly making false allegations.

An interesting topic for future research is to see whether an effective method for limiting false allegations can be devised based on (end-)user reputation via a so-called *reputation system* Lethin (2001); Cornelli et al. (2002). A reputation system (sometimes referred to as a *trust model*) records the reputation of, in this case, a user, as established from previous interactions between user and service or with other users. The service can use the reputation system to assign priority or trust to a user's request. An example of such a reputation system is the one used on the `slashdot.org` news and discussion Web site Malda (1999). A reputation system for users in the GDN would enable users to make allegations directly.

The correct operation of the GDN's scheme for limiting illegal distribution depends on two factors: (1) the goodwill of the producers and (2) the correct functioning of the access-granting organizations. In theory, the scheme works even if the majority of Internet users want to abuse the GDN for illegal distribution. Eventually, all abusers will have been black listed and only truthful people will have access. However, by the time we have reached this situation no person with truthful intentions will be making object servers available anymore. This scheme therefore practically depends on the goodwill of the producers. Given that their good name is at stake (the black list of abusers is public), we expect that most producers will behave.

The scheme itself provides some protection against misbehaving access-granting organizations. When a truthful access-granting organization mistakenly gives a previously blocked producer access again, an object server ends up serving illicit content. However, as before, this illicit content will be removed and its uploader is blocked when it is detected. When an access-granting organization (purposely or not) does not respond to accusations of abuse by producers it gave access to or (purposely) gives blocked producers access again, the AGO will get a bad reputation. Object-server owners will start refusing any producers the AGO accredited and eventually the AGO will be ousted from the GDN.

Although designed for supporting the cease-and-desist scheme, the GDN's architecture is also able to support other schemes for preventing illegal distribution. Content moderation could be supported by having access-granting organizations act as moderators and requiring users to upload content via their AGO. A reputation-based approach is also supported: an AGO could grant access to users based on their reputation rather than their willingness to show their passport and not appearing on the central black list, as in cease-and-desist.

4 Ensuring Authenticity

People downloading software from a software distribution network want to be assured of the authenticity and integrity of the software downloaded. In other words, is the package that they just downloaded the real thing or does it contain a malicious Trojan horse?

In the GDN, establishing the authenticity of software is the responsibility of the downloading user. In principle, the GDN guarantees only the integrity of the distributed software via the digital trace signature. It provides no authenticity guarantees other than the verified identity of the uploader as it appears on the trace certificate (which may be a pseudonym), as discussed above. Guarantees concerning the authenticity of software should therefore come from mechanisms outside the GDN. The GDN does, however, provide hooks for such external verification.

Currently, free software distributed via HTTP or FTP is authenticated using public-key cryptography (if at all). Maintainers of software packages digitally sign the files with a private key and publish the associated public key on the well-known Web site of the software package (e.g. `www.kernel.org` for the Linux kernel). People that download the software obtain the public key from the well-known Web site and use it to check the digital signature, thus establishing the authenticity of the software. We refer to this signature as the *end-to-end signature* to distinguish it from the trace signature introduced in the previous section. This authentication scheme requires that the associated public key is obtained from a trustworthy source that guarantees that the key actually belongs to the maintainer of the package. Note that even though Web sites currently do not meet this requirement they are nonetheless used for this purpose in practice. Furthermore, some of the end-to-end-signature schemes used do not protect against the renaming of files. Renamed files pose a risk because a malicious person may change the name of a file containing, for example, an old version of a software package with a known bug or security hole to that of the most recent version of that package.

The GDN supports only the automatic verification of end-to-end signatures. The GDN makes it the responsibility of the downloading user to obtain the proper public key. Concretely, when downloading a file from the GDN the associated end-to-end signature is downloaded along with it. The GDN client software then does the end-to-end authenticity check using a set of public keys supplied by the downloading user. If the set does not contain the required public key, the user is prompted to supply it.

Another result of the GDN's approach to authenticity is that the naming of files and DSOs in the GDN is not regulated. GDN producers are free to name their DSOs and files as they want. Downloaders should therefore not rely on names as an indication of what content they are downloading. They should make sure that they have the real name of the software they are looking for. The Globe Name Service is assumed to reliably translate Globe symbolic object names to the object handle of the object.

The reason for not having the GDN provide strong authenticity guarantees is that we expect users not to trust any statements a distribution network makes about the authenticity of the software they download. We expect users will want to verify themselves that the software they downloaded and which they will be running on their systems is what they expect it to be. Furthermore, it is also difficult for a distribution network such as the GDN to provide strong authenticity guarantees.

Consider the following example. To guarantee that the DSO named "GIMP 1.1.29" actually contains revision 1.1.29 of the GIMP application we would have to establish who is the maintainer of GIMP and make sure that only that person can create a DSO named "GIMP 1.1.29" in the GDN and can upload files into that object. Making sure that only a certain person can use certain names and edit certain objects is relatively easy, but establishing who is the maintainer of a specific package is, in general, rather difficult.

Note that although the use of digital signatures to provide end-to-end authenticity and integrity guarantees cancels the risk of malicious modification of a package after publication, it does not provide guarantees about the good intentions of the original publisher.

5 Guaranteeing Availability

A worldwide distribution network for free software should have around-the-clock availability. In this section we discuss the security measures taken to prevent external and internal attackers from disrupting the network. How the Globe Distribution Network deals with hardware and software faults (the other threat to availability) is discussed in Section 6.

To prevent external attackers from interrupting operation we enforce a role-based access control model Sandhu et al. (1996). The access-control model identifies principals, defines a number of roles and associated rights and how these roles are assigned. For example, for each distributed shared object there is a role named *replica* that can be assigned to an object server. This role enables an object server to host a replica of the DSO and advertise this replica as a representative of the object in the Globe Location Service (the service which clients use to locate replicas). The *replica* role is assigned to an object server by another replica. This access control model is not yet part of our prototype implementation of the Globe Distribution Network, as the Globe middleware's security facilities are still in the design phase Popescu et al. (2002). Instead, the prototype uses a simplified access control mechanism based on a Transport Layer Security (TLS) library Dierks and Allen (1999).

In addition to external attackers, we expect that GDN participants may attack the availability of the distribution network from the inside. This set of internal attackers includes both producers (i.e., software publishers) and object-server owners. Producers may abuse their rights to create objects and upload content to allocate excessive amounts of resources at object servers, thus making them unavailable to others. Object-server owners can stage a similar attack by abusing the replica rights assigned to their object server. In addition, owners may modify their object servers to act maliciously, for example, to serve other content than requested to downloaders, or have them confirm operations to clients and peers but not execute them.

The GDN supports a global and local (i.e., per object server) resource management system that impedes the over allocation of resources by internal attackers. The global resource management system, called the *GDN Quota Service (GDNQS)* limits the rate and annual number of distributed shared objects a producer can create. It is based on an observation from the free-software domain, in particular, that the rate at which new versions of a software package are published is fairly stable. Rarely more than one new version is published per day. Each producer is therefore assigned an annual quota of DSOs and cannot create more than a few DSOs per day. These quota are enforced by the GDNQS and the object servers. To create a new DSO the producer's upload tool first has to contact the GDNQS to obtain an *object-creation ticket*. Object servers participating in the GDN will create a new DSO only if the request is accompanied by such a ticket.

We keep a producer from allocating too many resources on a particular object server by introducing a local resource management system for object servers. The local resource management system keeps track of how many resources are used by each replica and to which producer this replica belongs, and denies allocation requests if a producer has already been allocated his fair share. In addition, we impose a limit on the size of the state of a distributed shared object (e.g. 1 Gigabyte), enforced by the code of the object's replica. The limit is set centrally for the whole GDN and is adjustable to allow growth in maximum file size. Furthermore, the local resource management system deletes replicas that are not frequently used, thus providing protection against producers and malicious object servers trying to reduce availability of the GDN by allocating useless additional replicas. An attacker could counter this measure by setting up clients that access the superfluous "malicious" replicas, thus keeping up their replicas' usage, but this requires a sustained effort from the attacker, and is therefore assumed unlikely.

The most basic attack for a malicious object server is to serve downloaders content other than what was requested. Doing so only hinders downloaders, as the integrity of the content is protected by the trace signature (see previous section). However, if the content served is not what the user expects but still traceable (i.e., a malicious object server could serve the user the content of a totally different object), users will not notice a problem until they do the end-to-end authenticity check. The fact that object servers may not be trustworthy makes the end-to-end authenticity check absolutely vital to the secure downloading of software from the GDN. Other attacks by malicious object servers, for example, attempts to modify the state of the object as held by other replicas are thwarted by the GDN's access control model. Replicas accept updates originating only from the object's core replicas, which run in trusted object servers (i.e., trusted by the GDN producer owning the object). In general, non-core replicas depend only on the core replicas (for state updates) and otherwise operate autonomously.

In addition to downloaders, object servers interact with two parties: producers (who request them to create new DSOs) and other object servers (which request them to create or delete replicas). To circumvent malicious object servers, downloaders, producers and object-server owners can specify which (other) object servers they trust or do not trust. At present, approval and disapproval of servers is specified via IP-address ranges and DNS domain names. In the future we hope to use a reputation system for object servers (see Section 3.3) to aid with server selection. The application of reputation systems to Globe applications is currently under investigation Pierre and Van Steen (2001).

6 Fault Tolerance

In this section we describe how the Globe Distribution Network is made fault tolerant. Fault tolerance has three aspects: availability, reliability and failure semantics. Availability indicates the probability of a system being available at any moment in time. The reliability of a system indicates how often it exhibits failure. Failure semantics define the state of the system after a failure Cristian (1991). Ideally, a system has strong failure semantics, implying that the system remains in a consistent state after the failure. An example of strong failure semantics are *atomic with respect to exceptions (AWE)* failure semantics, where either the operation is carried out, or it is not and the system is returned to the state it was in before the start of the operation. Reliability and strong failure semantics make a system easier to manage.

We first discuss the measures for ensuring availability and reliability of the GDN, after which we discuss the GDN's failure semantics. For a discussion of the fault tolerance aspects of the Globe middleware services, see, for example, Ballintijn et al. (1999).

6.1 Availability and Reliability

Making sure a distributed application is highly available and reliable starts, in principle, at the host and network level. Hosts and network can be made highly dependable using hardware redundancy, such as processors with a hot backup, disk arrays Chen et al. (1994), and multiple independent network connections. However, given the free nature of the Globe Distribution Network, we cannot employ hardware solutions to increase availability and reliability. We therefore start one level higher: making sure object servers are up and running most of the time.

6.1.1 Recovering Object Servers

Object servers can be made highly available by enabling them to quickly recover after a crash with most of their state intact. To this extent, Globe object servers currently support a simple checkpointing mechanism. Periodically, the object server creates a checkpoint by halting the processing of incoming requests, waiting until current requests have been processed, and then saving its state to disk. The state of an object server consists of the states of the replicas it hosts and the administration the object server maintains about these replicas. Once the object server's state is stable on disk, the previous checkpoint is deleted in an atomic disk operation. After a crash, the new object server reads the last complete checkpoint back from disk, recreates the replicas, and passes them their marshalled state. Each replica then reinitializes itself and synchronizes with its peers in an application- or even object-specific manner.

A GDN DSO's replica recovers from a server crash by contacting one of the DSO's (other) core replicas to see if its state is still current. If this is the case, the replica checks the integrity of the free software it stored on disk using the trace signatures of the files (see Section 3.2). Trace signatures are custom digital signatures that also contain a CRC for each block of the file to enable a fast integrity check. If the integrity check fails, the replica deregisters itself with the rest of the object and destroys itself. A replica currently also destroys itself when its state turns out to be out of sync, which is required to implement the GDN's failure semantics, discussed below.

Checkpointing the state of an object server in this fashion negatively affects the object server's availability as it does not process requests during a checkpoint. Fortunately, in case of the GDN, checkpointing time is low. First, none of the methods on a GDN DSO take much time to execute, so the checkpointing thread does not have to wait long before it can start checkpointing the server's state to disk after it has stopped the server from accepting new requests. Second, although the state of an object server used for GDN can be large, most of it is already stored

Table 1: Checkpoint and recovery times of a 15 Gigabyte object server.

Objects	Checkpointing (ms)	Recovery (ms)	Recovery w/o check (ms)
100	173	606,074	9,181
1000	1,281	653,480	16,297
5000	11,022	904,930	143,758

on disk, in particular, the software encapsulated by the DSOs. The checkpointing mechanism is such that this part of the object server's state need not be saved again, which considerably decreases the time to checkpoint. Our approach is also known as *user-directed checkpointing* Plank et al. (1995).

We tested our Java prototype implementation of the checkpointing mechanism on a dual-processor Pentium III at 933 MHz with 2 Gigabyte of memory and 10,000 RPM Ultra-160 SCSI disks, running RedHat 7.1 and using IBM's Java Developer Kit (see Section 7). In this test, the object server was idle (i.e., not processing client requests) and had 15 Gigabyte of total state. We divided the 15 GB of state over 100, 1000 and 5000 objects and measured checkpointing time, recovery time with and without integrity checking in these three cases. The results of the test are shown in Table 1 and are the average of at least three runs.

The table shows that checkpointing time is indeed low for the object server. If the server checkpointed every hour while being idle, its availability in the absence of failure would be between 99.69 % (with 5000 objects) and 99.99 % (with 100 objects). Recovery time after a failure is considerable due to the objects checking the integrity of their state. Our prototype of the checkpointing mechanism has, however, not yet been fully optimized, and therefore currently restarts only one replica at a time after a crash. As a result, not all capacity of the machine was used during the test. The recovery rate of our prototype currently also decreases with the number of objects in the server, from 24.7 MB/s with 100 objects to 16.6 MB/s for 5000 objects, indicating there is more room for improvement. In principle, the recovery rate should remain stable as it is disk I/O bound. The recovery times listed in the table are minimum values as they do not include the time necessary for the replicas to verify the consistency of their state with a core replica. The duration of this verification depends on the latency of the network path between replica and core, as it currently involves setting up a TCP connection to the core replica and doing a single remote procedure call.

6.1.2 Object Redundancy

As explained in Section 2, the distributed shared objects of the GDN replicate themselves over the set of object servers to make the software they encapsulate efficiently available to the clients. This replication for performance naturally increases the availability of the DSOs in the GDN. If the replica nearest to the client is down, the client will connect to another nearby replica. This natural redundancy does not apply to all software packages, however. As the replication degree depends on the number of clients, unpopular software packages may not have any extra replicas. To provide a minimum level of fault tolerance, each DSO therefore always has two or more core replicas.

6.2 Failure Semantics

To remain manageable, the Globe Distribution Network should provide "atomic with respect to exceptions" failure semantics, that is, an operation is carried out, or it is not and the distribution network is returned to the state it was in before the start of the operation. Providing such semantics for downloads is easy, as downloads in the GDN are stateless (i.e., distributed shared objects do not keep track of downloads) for scalability reasons. At present, we do not provide these failure semantics for uploads, as uploads are operations that consist of multiple method invocations on a DSO (see Section 2.1), and the Globe middleware does not yet provide a transaction mechanism to atomically execute such sequences.

Instead, the GDN strives to make uploads succeed whenever possible. It sacrifices replicas that may be just temporarily disfunctional in order to prevent having to report failure, under the assumption that replicas will be recreated by the object if client demand requires it. This solution is considered sufficient for the time being, as the number of uploads into a GDN DSO is low. The number of uploads is low due to the limited amount of software that is placed in an individual DSO (i.e., we place each new revision of a software package in a separate DSO). The details of this ad-hoc solution are discussed in Bakker (2002).

Above we described how the GDN relies on digital signatures to guarantee the integrity of a file distributed through the GDN. This integrity check also detects any data corruption that has occurred due to failures inside the GDN that may have gone unnoticed. In this sense, the trace signatures on files in the GDN provide *end-to-end* integrity protection, a desirable property Saltzer et al. (1984).

7 Performance

We have conducted a number of experiments to test the performance of our prototype implementation of the Globe Distribution Network. In the first two experiments we compare the performance of a single Globe object server to that of the Apache HTTP server The Apache Software Foundation (2002). In the last experiment we provide evidence that that download time in the GDN can indeed be improved by using a nearby server. We discuss each of these experiments in turn.

7.1 Experiment 1: Single Replica Performance

The first experiment is aimed at measuring the average throughput per client for a large number of clients simultaneously downloading the same file. In this experiment with the Globe object server, clients download a 30 Megabyte file from a single distributed shared object. The DSO consists of a single replica running in the object server. The measured per-client throughput is compared to downloads of the same file from an Apache HTTP server.

We used the following hardware and software. The client machines were dual-processor Pentium III at 1 Gigahertz with 1 Gigabyte of memory, running RedHat 7.2. The server machine was a dual-processor Pentium III at 933 MHz with 2 Gigabyte of memory and 10,000 RPM Ultra-160 SCSI disks, running RedHat 7.1. All machines ran custom configured kernels, and were connected via a full-duplex 100 Mb/s Ethernet. For the tests with Apache, we used Apache version 1.3.19, and the `wget` HTTP client version 1.6. The Apache server was configured following advice from RedHat Likins (2002). For the tests with GDN, we used version 1.0 of the GDN implementation (available from `http://www.cs.vu.nl/globe`) which is written in Java. To execute the Java code we used the IBM Developer Kit for Linux, Java 2 Technology Edition, version 1.3-2001-06-26, which includes a high-performance just-in-time (JIT) compiler and is available from `http://www-106.ibm.com/developerworks/java/jdk/linux130/`.

For the measurements with 1–50 concurrent clients each node ran a single GDN or HTTP client. For the measurements with 60–100 concurrent clients, each node ran 2 client programs, one on each CPU. In this experiment, we focused on network throughput and, as a result, the download time measured at the client (from which we calculate the throughput) does not include the time used accessing the Globe Name Service or Globe Location Service, nor the time required to check the trace signature of the file after download (i.e., we did not measure end-to-end performance). The GDN client connects directly to the replica based on a stored contact address (see Section 2.1). Both the HTTP client and the GDN client discard the downloaded data by writing it to `/dev/null`, so there is no disk I/O at

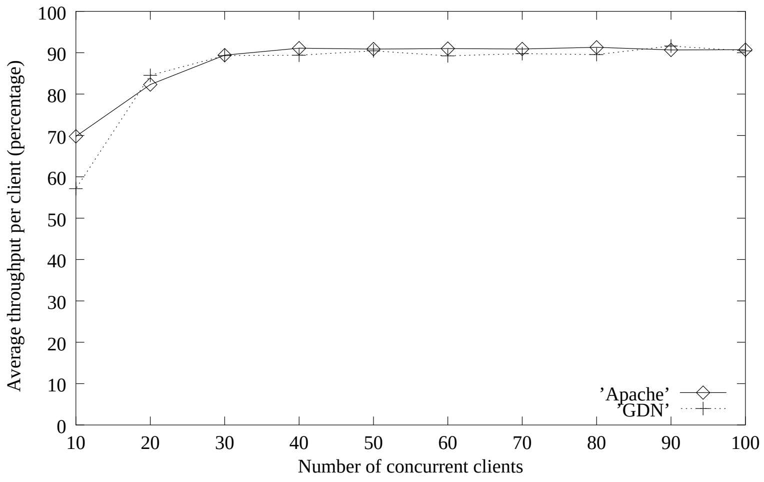

Figure 4: Average throughput per client for Apache and the GDN as a percentage of TCP's maximum theoretical throughput over 100 Mb/s Ethernet, measured using 1–50 dual-processor client machines.

the client side. GDN clients download the file in blocks of 1 Megabyte.

The results of this experiment are shown in Figure 4. It shows the average throughput per client for 1–100 concurrent clients compared to the maximum theoretical throughput of TCP over 100 Mb/s Ethernet (which is 11.3 MB/s). We repeated the experiment three times for both types of servers. The numbers used to construct the figure are the average of the three runs. As the figure shows the performance difference between Apache and the Globe object server is less than 2 percentage points for large numbers of clients. The performance bottleneck is the 100 Mb/s Ethernet network.

7.2 Experiment 2: Performance with Multiple Replicas

In the first experiment, the clients all downloaded large files from the servers. In the second experiment we compared the servers under a heterogeneous workload. Both servers were loaded with the 50 most popular files on the SourceForge free-software site in October 2001. In the GDN case, each file was placed in a separate distributed shared object. The size of these files ranged from 21 KB to 15 MB (average 1.5 MB). At the client side 50 clients were started, and each client continuously downloaded the same file from the server. After 30 minutes all clients

Table 2: Results of Experiment 2.

Server	Downloads per second	Kilobytes per second
GDN	15.66	9486.75
Apache	19.65	9942.59
Difference	+25.47%	+4.81%

were killed and the total number of successful downloads was counted. The same hardware and software setup was used as in the previous experiment.

The results of Experiment 2 are shown in Table 2. The experiment was repeated three times, and the numbers shown are the average of the three runs. In terms of the number of files served, Apache outperforms the GDN by more than 25%. In terms of number of bytes served, however, Apache is just 4.81% better than the GDN object server. The higher number of files for Apache was due to the fact that the GDN created a new proxy of the object for each download and is generally slower at downloading small files due to overhead, as further study revealed.

7.3 Experiment 3: Download Time and Proximity

Two important properties of the Globe Distribution Network are that:

1. it allows clients to download a copy of the software from a nearby server

2. it *binds*, that is, locates and connects to this server using *proportional communication*. Proportional communication means that the client does not send any wide-area binding messages if the replica is on the local network.

The second property is mainly due to the Globe Location Service which has lookup costs proportional to the distance between lookup requester and nearest replica (see Section 2.1).

These properties can improve both download time and scalability as they avoid communicating over shared wide-area networks where bandwidth is assumed to be scarce and latency high due to geographical distance and possibly congestion. To evidence that download time in the GDN can indeed be improved by using a nearby server and that the time for locating a replica is smaller when the replica is closer by, we ran the following experiment.

We used a machine located in San Diego, CA to download a one Megabyte file from two locations: Ithaca, NY (approximately. 4000 kilometers away) and Amsterdam, The Netherlands (approximately. 9000 kilometers away), and measured the download times. The download time is subdivided into three components. The

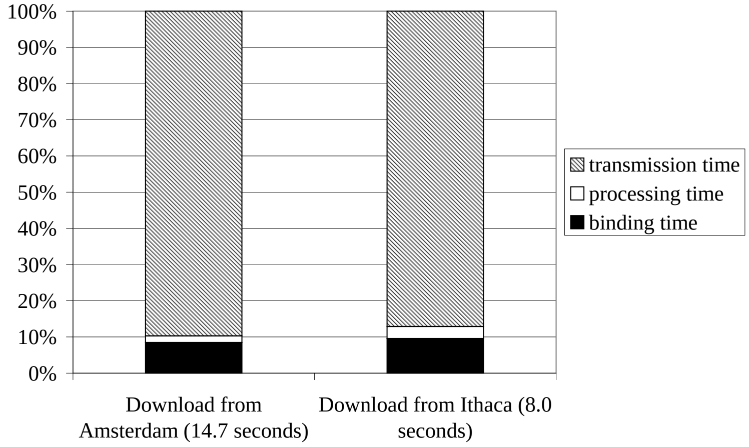

Figure 5: Histogram of the download times from Amsterdam and Ithaca, broken down in three components.

first component, *binding time*, entails the time spent in the first part of the download where the client establishes a connection to the replica, by contacting the Globe middleware services and creating a proxy for the object in its local address space. The second component, *transmission time*, is the total time spent on the network during the download, and the third component, *processing time* is the time used by the client's proxy and the replica to process the client's download request.

The two download times and their components are shown in Figure 5. The numbers used are the average of 99 downloads from both locations. In this experiment, the download to San Diego from the closer server in Ithaca is indeed faster than the download from Amsterdam (by 6.7 seconds). The histograms also shows that in this experiment binding time is decreased considerably when binding to a replica which is nearer. The percentage of total download time spent binding only slightly increases (from 8 to 10 % of the total download time) when the replica in Ithaca is used. This percentage would have been considerably higher if binding time did not decrease with client–replica distance. A more detailed breakdown of the download times and a comparison to downloads via HTTP can be found in Bakker (2002).

8 Conclusions

The Globe Distribution Network (GDN) is an application for the efficient distribution of freely redistributable software packages. It has been developed as a test application for a new middleware platform called Globe, which is designed to facilitate the development of large-scale Internet applications. Distribution of the free software is made efficient by encapsulating the software into Globe distributed shared objects and having efficiently replicating the objects near to the clients downloading the software. Replication of the software is automated because distributed shared objects manage their replication themselves based on past and present client demand. The GDN guarantees its availability despite attacks by out– and insiders

Instead of doing content moderation at upload time to prevent the illegal distribution of copyrighted material or other illicit content, the Globe Distribution Network takes a novel approach where publishers are given direct access to the network. In this optimistic approach all content uploaded into the network is made traceable to its publisher (by means of digital signatures) allowing illicit material to be removed from the GDN immediately after it is found and the publisher of this material to be banned from the GDN. The Globe Distribution Network exploits the replication of the software to achieve high availability and has well-defined failure semantics when failures can no longer be masked.

The performance of our prototype implementation of the GDN is adequate and it will serve as a framework for validating our ongoing research into adaptive replication protocols in the future. The source code for both the Globe Distribution Network and the Globe middleware platform are freely available under the BSD software license at `http://www.cs.vu.nl/globe`.

Acknowledgments

We would like to thank Chandana Gamage, our staff programmers, Patrick Verkaik and Egon Amade and our sponsor, Stichting NLNet for their support in the development of Globe and the Globe Distribution Network.

References

AGHA, G., Ed. 2002. *Communications of the ACM: Special Section on Adaptive Middleware*. ACM.

BAKKER, A. 2002. An Object-Based Software Distribution Network. Ph.D. thesis, Division of Mathematics and Computer Science, Faculty of Sciences, Vrije Universiteit, Amsterdam, The Netherlands.

BAKKER, A., AMADE, E., BALLINTIJN, G., KUZ, I., VERKAIK, P., VAN DER WIJK, I., VAN STEEN, M., AND TANENBAUM, A. 2000. The Globe Distribution Network. In *Proceedings 2000 USENIX Annual Technical Conference (FREENIX track)*. San Diego, CA, USA, 141–152.

BAKKER, A., VAN STEEN, M., AND TANENBAUM, A. 2001. A Law-Abiding Peer-to-Peer Network for Free-Software Distribution. In *Proceedings IEEE Symposium on Network Computing and Applications (NCA'01)*. IEEE Computer Society, Cambridge, MA, 60–67.

BALLINTIJN, G., VAN STEEN, M., AND TANENBAUM, A. 1999. Simple Crash Recovery in a Wide-Area Location Service. In *Proceedings 12th International Conference on Parallel and Distributed Computing Systems*. Fort Lauderdale, FL, USA, 87–93.

BALLINTIJN, G., VAN STEEN, M., AND TANENBAUM, A. 2001. Scalable User-Friendly Resource Names. *IEEE Internet Computing 5,* 5 (Sept.), 20–27.

BURK, D. 2001. Copyrightable Functions and Patentable Speech. *Communications of the ACM 44,* 2 (Feb.), 69–75.

CARMODY, P. 2002. The world's first illegal prime number? `http://asdf.org/~fatphil/maths/illegal1.html`.

CHEN, P., LEE, E., GIBSON, G., KATZ, R., AND PATTERSON, D. 1994. RAID: High-Performance, Reliable Secondary Storage. *ACM Computing Surveys 26,* 2 (June), 145–185.

CORNELLI, F., DAMIANI, E., DE CAPITANI DI VIMERCATI, S., PARABOSCHI, S., AND SAMARATI, P. 2002. Choosing Reputable Servents in a P2P Network. In *Proceedings 11th International World Wide Web Conference*. Honolulu, HI, USA.

CRISTIAN, F. 1991. Understanding Fault-Tolerant Distributed Systems. *Communications of the ACM 34,* 2 (Feb.), 56–78.

DIERKS, T. AND ALLEN, C. 1999. RFC 2246: The TLS Protocol Version 1.0.

FREE SOFTWARE FOUNDATION, INC. 1991. GNU General Public License Version 2. `http://www.fsf.org/licenses/gpl.txt`.

KUZ, I., VAN STEEN, M., AND SIPS, H. 2002. The Globe Infrastructure Directory Service. *Computer Communications 25,* 9 (June), 835–845. Elsevier Science, Amsterdam, The Netherlands.

LETHIN, R. 2001. Reputation. In *Peer-to-Peer: Harnessing the Power of Disruptive Techn ologies*, A. Oram, Ed. O'Reilly and Associates, Sebastopol, CA, USA, Chapter 17, 341–353.

LIKINS, A. 2002. System Tuning Info for Linux Servers. `http://people.redhat.com/alikins/system_tuning.html`.

MALDA, R. 1999. Slashdot Moderation. `http://slashdot.org/moderation.shtml`.

NEUMAN, B. C. 1994. Scale in Distributed Systems. In *Readings in Distributed Computing Systems*, T. Casavant and M. Singhal, Eds. IEEE Computer Society.

NIELSEN, J. 1995. *Multimedia and Hypertext: The Internet and Beyond*. AP Professional, Boston, MA, USA.

ORAM, A., Ed. 2001. *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly and Associates, Sebastopol, CA, USA.

PIERRE, G. AND VAN STEEN, M. 2001. Globule: a Platform for Self-Replicating Web Documents. In *Proceedings 6th International Conference on Protocols for Multimedia Systems*. Enschede, The Netherlands.

PLANK, J., BECK, M., KINGSLEY, G., AND LI, K. 1995. Libckpt: Transparent Checkpointing under Unix. In *Proceedings USENIX Winter 1995 Technical Conference*. New Orleans, LA, USA, 213–223.

POPESCU, B., VAN STEEN, M., AND TANENBAUM, A. 2002. A Security Architecture for Object-Based Distributed Systems. In *Proceedings 18th Annual Computer Security Applications Conference*. Las Vegas, NV, USA, 161–171.

SALTZER, J., REED, D., AND CLARK, D. 1984. End-To-End Arguments in System Design. *ACM Transactions on Computer Systems 2,* 4 (Nov.), 277–288.

SANDHU, R., COYNE, E., FEINSTEIN, H., AND YOUMAN, C. 1996. Role-Based Access Control Models. *IEEE Computer 29,* 2 (Feb.), 38–47.

SIMONS, B. 2000. From the President: To DVD or Not To DVD. *Communications of the ACM 43,* 5 (May), 31–32.

THE APACHE SOFTWARE FOUNDATION. 2002. The Apache HTTP Server. `http://www.apache.org/`.

UNITED STATES GOVERNMENT. 1998. Digital Millennium Copyright Act. United States Public Law No. 105-304.

U.S. DEPARTMENT OF COMMERCE. 2001. Commercial Encryption Export Controls. `http://www.bxa.doc.gov/Encryption/Default.htm`.

VAN STEEN, M., HAUCK, F., AND TANENBAUM, A. 1998. Locating Objects in Wide-Area Systems. *IEEE Communications Magazine*, 104–109.

VAN STEEN, M., HOMBURG, P., AND TANENBAUM, A. 1999. Globe: A Wide-Area Distributed System. *IEEE Concurrency 7,* 1 (Jan.), 70–78.

WORLD INTELLECTUAL PROPERTY ORGANIZATION. 1996. WIPO Copyright Treaty. In *WIPO Diplomatic Conference on Certain Copyright and Neighbouring Rights Questions*. Geneva, Switzerland. `http://www.wipo.int/treaties/ip/copyright/index.html`.