A Wide-Area Distribution Network for Free Software

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

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 to its availability. By exploiting the replication of the software and using fault-tolerant server software, the Globe Distribution Network achieves high availability. A prototype implementation of the GDN is available from `http://www.cs.vu.nl/globe/`.

Categories and Subject Descriptors: C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications*; H.3.5 [**Information Storage and Retrieval**]: Online Information Services—*Data sharing*; K.4.1 [**Computers and Society**]: Public Policy Issues—*Abuse and crime involving computers*; *intellectual property rights*

General Terms: Design, Legal Aspects, Management, Performance, Security, Reliability

Additional Key Words and Phrases: Distributed objects, middleware, software distribution, file sharing, traceable content, copyright, wide-area networks

1. INTRODUCTION

The scale of distributed applications can be 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

Author's address: A. Bakker, Department of Computer Science, Faculty of Science, Vrije Universiteit, De Boelelaan 1081a, 1081 HV, Amsterdam, The Netherlands; email: arno@cs.vu.nl.

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, particularly, 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 (extrafunctional) aspects and required techniques in a comprehensive manner. In addition to comprehensiveness, flexibility is particularly important for a development platform. 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 extrafunctional aspects such as replication 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 extrafunctional 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. What complicates matters is our design goal which is to allow a group of volunteers to operate the GDN: the server and network resources should be donated and the application should be managed by volunteers, as in the current FTP-based infrastructure. These people cannot be expected to vet content that thousands of content providers wish to distribute via GDN.

Our approach to inhibiting illegal distribution is therefore 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 after the publication has proven illegal. This optimistic approach is necessary to minimize the amount of work for the volunteers. 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. A prototype of the GDN implementing a considerable part of these features can be downloaded from `http://www.cs.vu.nl/globe/`.

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. In Section 3, we present our approach to preventing illegal distribution. Section 4 describes how we guarantee the availability of the GDN despite attacks by insiders and outsiders, and Section 5 presents the fault tolerance measures of the GDN. Related work is discussed in Section 6. We conclude in Section 7.

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 that is being distributed. The distributed shared objects are replicated over a collection of user-level *object servers* located throughout the Internet. The clients, distributed shared objects, and object servers are supported by a number of middleware services. The architecture of GDN and Globe is described in detail in Bakker et al. [2003].

2.1 Downloading Software

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 (binary) identifier. Second, the client resolves the object handle of the object to a *contact address* of a replica of the object. The contact address contains the network address of the replica and information on how to contact it (i.e., which

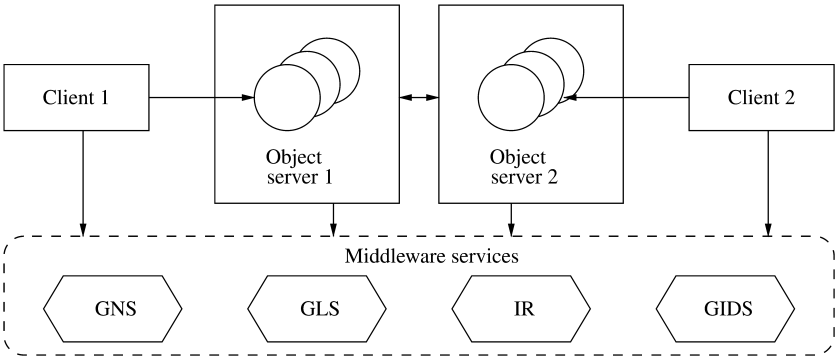

Fig. 1. The architecture of the Globe Distribution Network. A circle represents a replica of a distributed shared object; a rectangle represents a machine; a diamond represents a service, implemented by one or more machines; a dashed rounded box represents a collection.

protocols to use in communication). In particular, 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 the lookup requester and the 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). In the final step, 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 `startFileAddition`, `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.

To the end user, downloading software is similar to downloading files from a FTP or Web site. The GDN supports an HTTP-to-GDN gateway that allows Web browsers to directly download files from distributed shared objects. For uploading software into the GDN, we have a FTP client-like tool that simplifies the regular tasks of a software publisher, handling the interaction with objects, object servers, and middleware services.

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 five subobjects as illustrated in Figure 2. The *replication subobject* (labeled R in the figure) contains the

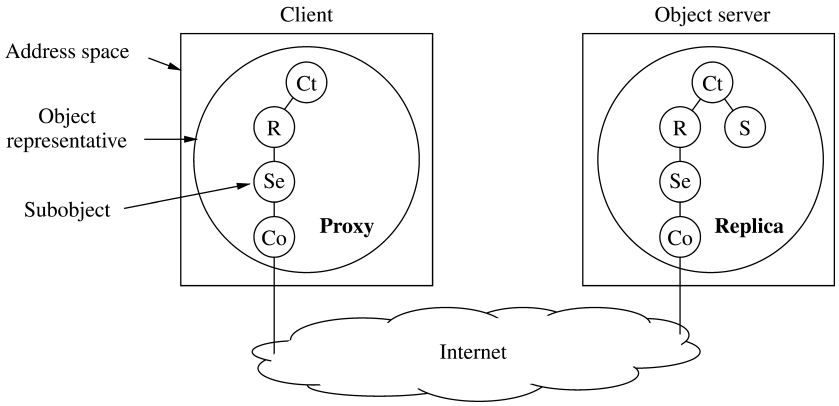

Fig. 2. The structure of distributed shared objects.

implementation of the replication protocol used by this object. The *security subobject* (labeled Se in the figure) implements the object's security policy and protocols used. The *communication subobject* (Co in the figure) satisfies the replication and security subobjects' 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 fault-tolerance aspects.

The modular structure enables the application developer to change the subobjects that handle replication or other extrafunctional 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.

Our Java prototype implementation of Globe includes robust implementations of all the Globe middleware services and run-time system, a number of replication protocols, and a security subobject that uses multiple security protocols.

2.3 Efficient Distribution via DSOs

In the GDN, the distributed shared objects use a replication protocol 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 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 reports these to the trusted *core replicas* which coordinate global operations (such as state updates) in the object. The core replicas determine which regions are generating many requests and place new replicas there. Replicas not only report 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]. In such an event, they can autonomously create extra replicas in the busy regions.

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 GDN and can locate an object server with specific properties in a particular region of the Internet [Kuz et al. 2002]. To this extent, the GIDS divides the world into a set of base regions (typically the size of a LAN), which are organized into a hierarchy currently based on their geographical location. The location of clients and object servers are expressed as base regions in this hierarchy.

Full details on the (active) replication protocol, such as how it balances load, can be found in Bakker [2002]. We have implemented a simplified version with a single core replica in our GDN prototype. Our research into intelligent protocols is continued in the Globule project [Pierre et al. 2002; Sivasubramanian et al. 2003].

3. ILLEGAL DISTRIBUTION

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 permit free distribution amongs other things. A well-known license is the GNU General Public License (GPL) [Free Software Foundation, Inc. 1991].

The preceeding 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 because 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. The GDN does not at present address these national differences as these issues require further investigation. In the meantime, we define our own global policy of what can be distributed via

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

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 GDN 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 who put it there [United States Government 1998]. A similar risk exists for pornography and other illicit materials. If there is a risk of liability, people and organizations may not volunteer their time and resources to the GDN. This conflicts with our design goal of wanting 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.

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 uploaded. Manual checking is required because a computer cannot tell copyrighted from noncopyrighted content or illicit from legal content.

We deemed content moderation inappropriate for the GDN. Finding a group of moderators that are willing to devote their time to tedious and time consuming work will probably be hard. Also, content moderation introduces a (potentially long) delay between the initial submission for publication and the actual publication in the distribution network. Finally, the moderators may be legally liable for inadvertently approving copyrighted material.

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 also has to create a large user base such that the reviewers start considering the software as a candidate for replication. Unfortunately, this simple approach 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 employed in the GDN as explained in Section 3.3.

3.1.3 *Cease-and-Desist.* Our solution to preventing illegal distribution in the GDN 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 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. The cease-and-desist scheme tries to limit illegal distribution by banning provably malicious users from the distribution network.

This scheme 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].

The 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.

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, since there is no possibility to keep out all illegal content in any scheme. Even content moderation, which is the best scheme for limiting the amount of illegal content, cannot filter out all illegal content. The reason is that detecting illegal content requires manual checks which are error prone and, furthermore, may be defeated by cleverly encoding illicit content into inconspicuous content, a process called steganography [Katzenbeisser and Petitcolas 1999]. A powerful example supporting this latter argument is the so-called "first known illegal prime number" [Carmody 2002], which when

represented as bytes is a GZIP-ed version of the DeCSS source code. The DeCSS source code can be used to circumvent copyright protection on digital video discs.

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 blacklist when the user is found guilty of illegal distribution. This blacklist is checked at each application for access to the distribution network, thus keeping out violators reapplying for access.

The advantage of the cease-and-desist scheme over content moderation is that the amount of work for moderators is small when the level of abuse is low. In this 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 more effective when abuse is high. To achieve the laborefficient situation when abuse is low, there should be few false reports which is the third condition specified.

False reports are not expected to be a problem. At present, in the United States, it is the responsibility of the copyright holder to report the illegal distribution of his content to those managing the violating servers. In case of the GDN, the copyright holders would report to the group of moderators via an official procedure who would then determine the legitimacy of the complaint and take action. Likewise, it could be the legal responsibility of law enforcement agencies to report illicit content. In both cases, reporting procedures are expected to discourage false reports.

For completeness, the cease-and-desist scheme also addresses the case where the users of the distribution network themselves are expected to regulate their own network. In that exceptional case, false reports do pose a problem. Malicious people can try to undermine the prevention scheme by swamping the moderators in unnecessary work. We argue that to keep the number of false reports at bay, there must a threshold for a user to submit a report.

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 (AGO) verifies the candidate's identity by checking his passport or other formal means of identification. In addition, the organization checks if this person has been banned from the GDN by any of the other AGOs by consulting a central blacklist.

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

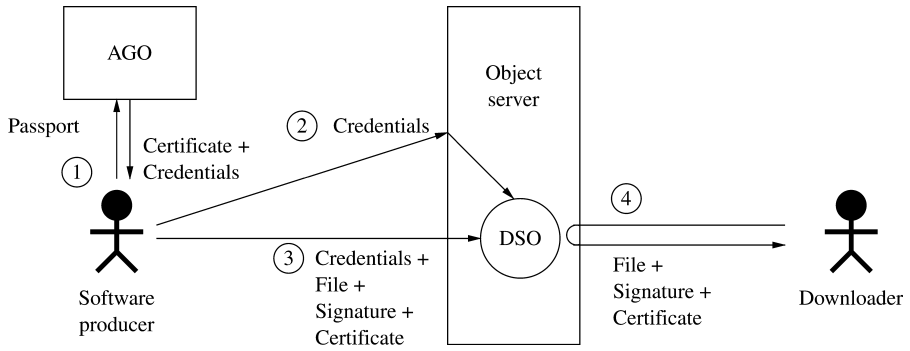

Fig. 3. Basic operation of the GDN with traceable content.

digitally signs this certificate. This certificate is called the *trace certificate* and the key pair of which the public key on the 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. As an AGO also processes reports of illicit content, it is different from certification authorities like VeriSign. An AGO can, however, choose to accept certificates by certain CAs as a proper means of identification.

An owner of an object server specifies which producers it wants to allow to access 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 blacklist 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.* The application and upload procedure is shown in Figure 3. In Step 1, a software producer identifies himself to an AGO and receives a trace certificate and Globe credentials in return. In Step 2, the producer requests an object server that trusts the AGO that the producer got his credentials from to create a distributed shared object (arrow 2). Next, the producer creates a digital signature for the file to be published 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 (arrow 3 in Figure 3).

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. Multiple files can be uploaded into a single DSO using this procedure. Finally, a user downloads the file, the trace certificate and the trace signature from the DSO, and subsequently verifies that they match (arrow 4).

3.2.2 *Blocking Illegal Publishers.* To ban a producer from the GDN when a copyright holder or law enforcement officer finds illicit content that is traceable to that producer, the following procedure is executed. The finder notifies all object-server owners and the access-granting organization that gave the suspected producer access for 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 blacklist shared by all AGOs. In effect, the violator is thus banned from the GDN. In the case of self regulation, a regular user that finds illicit content in the GDN must contact a GDN producer who will make the accusation on his behalf. It is assumed that governments will not make false allegations (e.g., through their law enforcement officers).

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 accuser 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 local jurisdiction. 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 system-wide 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 by the moderators. This policy provides protection against malicious producers trying to remove well-known software packages from the GDN.

3.2.3 *Appeals Process.* There are three potential problems with access-granting organizations.

(1) Due to sloppiness or forged identification means, an AGO may grant a trace certificate to the wrong person.
(2) A malicious or sloppy AGO may blacklist the right person for the wrong reason.
(3) An AGO may refuse to blacklist a person, although the software he published is illegal to distribute.

The first problem is solved by introducing an appeals process. When a person A discovers that a trace certificate has been handed to the wrong person B, and B has not yet been placed on the central blacklist, A contacts the AGO that issued the certificate. When B has abused its rights and is blocked, A contacts the *GDN Administration*. The GDN Administration is the central authority that governs the GDN, and, in particular, controls which organizations can be access-granting organizations for the GDN. In both cases, the appeals committee of either AGO or GDN Administration does an extra thorough check of the

identity of the appealer. If the appealer shows its identity to satisfaction, the appeals committee attempts to contact the current holder of the trace certificate. To this extent, when a producer registers with an AGO, he also supplies a contact address. If the holder cannot be contacted, the original certificate is revoked and a new one is issued to the appealer. If the holder is reachable, he will be asked to reidentify himself. Assuming any fraud will be revealed when comparing identification means, the appealer or the holder will be granted a (new) trace certificate. We consider sloppiness the main cause of this problem as we assume formal identification means are not easily forged. To prevent identity theft, formal identification is necessary and weaker authentication schemes are deemed unsuitable.

If identity theft because of sloppy AGOs becomes too frequent, the GDN can delegate the AGO's function of establishing identity to well-known Certification Authorities such as VeriSign. AGOs would then only have to issue trace certificates and Globe security credentials. An additional advantage of this solution would be the costs associated with obtaining a CA-certified identity certificate since this creates a threshold for malicious persons wanting to join. This threshold may also deter legitimate producers, but we expect that people will appreciate the properties of the GDN (such as automatic fail-over) and that those people who really need a more powerful distribution channel will not be deterred. Experience in the Debian Project shows that software publishers will go to great lengths to join a convenient publication system (see Section 6).

The second problem is also solved by allowing appeals. A person blacklisted incorrectly can ask the GDN Administration to reinvestigate the accusation. The AGO involved is required to show the evidence to the GDN Administration. This possibility for appeal can also be used in cases where initially illegal software is ruled to be legal at a later time (e.g., the DeCSS code [Simons 2000]). If an AGO refuses to blacklist a certain software publisher (the third problem), the distribution of his illegal software will cease automatically as the object-server owners will no longer allow producers from that AGO on their systems.

We offer no solution to the problem of stolen trace key pairs. Producers who fear their keys are at risk from viruses or worms should take appropriate measures to protect them such as creating trace signatures and object-pedigree chains (see Section 4) on a separate, non-networked computer.

Our GDN prototype contains a complete implementation of cease-and-desist which is described in Bakker [2002, Chapter 5].

3.3 Discussion

The cease-and-desist scheme is an optimistic scheme that assumes that the majority of producers request access to the GDN to legitimately publish their software. If this assumption does not hold, the GDN can switch to a reputation-based approach, as its security architecture is flexible. For example, individual object-server owners can let specific producers get access to their system. Other owners may delegate the access-control task to an AGO that allows only well-known publishers. The strength of the security architecture, whichever scheme is used, is that it forces content to be traceable so that, when producers go

astray, at least there is a simple way of removing all disputed content (unlike the current Internet).

3.3.1 *General Policy Variations.* Blocking an illegal publisher for life may seem like a harsh policy. It may be possible to adopt weaker policies where access is regranted after some period of time. One could even consider blacklisting just a particular software package with the threat of blacklisting the producer if he persists. Whether or not weaker policies are possible depends on the frequency of abuse and the law. If there are many people abusing their rights, allowing them to reenter is not a good idea. The law may also require a lifetime block in some cases (e.g., child pornography). We choose to be hard on violators in order to keep the volume of illegal content down.

3.3.2 *Self Regulation Policies.* In the exceptional case where the users of the GDN have to be self-regulating, the number of false reports must be kept low for cease-and-desist to work as intended. Our initial proposal is to create a threshold by requiring that allegations are made by or through an active producer that will be blocked if the allegation proves false. Blocking the accuser is harshs and may encourage entrapment of an accusing producer by someone else. On the other hand, making sure that bad content and its publishers are removed by making sure proper allegations are not lost in a flood of false reports is vital for the legitimacy of the distribution network. If there is too much illicit content, the network may be forbidden.

Our initial policy of immediate and permanent blockage of a false accuser may be too conservative. Less conservative policies, such as blocking a producer only after a few mistakes or revoking access for just a period of time, should be tried in practice to see if the number of false reports remains acceptable. Counting the mistakes a producer makes is an example of a simple *reputation system*. Reputation systems are a promising new development [Lethin 2001; Cornelli et al. 2002], and we intend to investigate their potential for the GDN. A particularly interesting topic for future research is to see whether an effective method for limiting false allegations can be devised based on end-user reputation which would allow end users to report without intermediaries. Such an end-user reputation system would represent another type of threshold for allegations. Other possible thresholds are requiring a refundable fee to be deposited with the accusation or requiring an accusation to be endorsed by multiple producers.

One can also argue that immediate and permanent blockage is not conservative enough. As anyone is allowed to become a producer, people could sign up with the sole intention of making false reports and not publishing software. As indicated earlier, cease-and-desist is an optimistic scheme that does not hold up to such active and massive malicious behavior, and a better choice is a reputation-based approach.

Given the right sanctions policy, we do not expect that finding a willing producer is a problem. 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. Finally, we believe that self-regulation is

not a situation that should be encouraged. Making copyright holders and law enforcement responsible for submitting allegations is better because it is also more in line with most legal systems.

3.4 Ensuring Authenticity

People downloading software from a software distribution network want to be assured of the authenticity and integrity of the software downloaded. In 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 as discussed previously. Guarantees concerning the authenticity of software should therefore come from mechanisms outside the GDN, such as *end-to-end digital signatures* which is a common practice with software distribution today.

4. 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 GDN deals with hardware and software faults (the other threat to availability) is discussed in Section 5.

To prevent external attackers from interrupting operations we enforce a role-based access control model [Sandhu et al. 1996]. The access-control model identifies principals and 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.

The GDN prototype uses the Globe security framework to implement this access-control model [Popeseu et al. 2002, 2003a, 2003b]. In this framework, each distributed shared object has a public/private key pair. This key pair is linked to the object by including an SHA-1 digest of the public key in the object's unique object identifier (the object handle, see Section 2.1). The holder of this key pair, known as the *object owner*, uses it to delegate specific permissions to other key pairs by means of *authorization* and *administrative certificates*. An authorization certificate specifies which methods of a distributed shared object the holder of a key pair is allowed to invoke and which replication and security control messages it is allowed to send. An administrative certificate specifies what authorization certificates the holder is allowed to issue and serves as a parent certificate for authorization certificates. The object's public key is placed in a self-certified administrative certificate that is at the root of the delegation hierarchy from object owner to end-user authorization certificates that is created by the administrative certificates.

In the GDN, when the object owner creates the core replicas of the object, it creates a key pair, an authorization certificate, and an administrative certificate for each replica. The authorization certificate enables the core replicas to, for example, disseminate updates to the state of the object to other (noncore) replicas. The administrative certificate allows them to grant certificates to noncore replicas that are created to handle increased client demand. A noncore replica will also receive an authorization and an administrative certificate. The former allows it to identity itself as a replica of the object to clients and core replicas. The latter allows it to delegate similar rights to other noncore replicas that it creates in the event of a flash crowd. End users, in general, will also authorize their actions via certificates, in particular, if they are allowed to modify the state of the object (e.g., upload software). Objects can, however, also allow unauthorized (i.e., certificateless) users to invoke methods, for example, just their read-only methods. The clients and replicas use various protocols for authentication and secure communication. In our prototype, we currently use TLS [Dierks and Allen 1999] and a newly designed protocol [Crispo et al. 2004].

The authorization certificate scheme can be linked to identity certificates to allow access control based on identity. Concretely, for the GDN, a producer will create a pedigree certificate chain for each object he creates, linking his trace (identity) certificate to the root administrative certificate for the object. When object servers are asked to create replicas for the producer's object, the pedigree chain is sent along in the request. This enables the object servers to check if the object belongs to a producer that was certified by an access-granting organization they are configured to trust.

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 content 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 overallocation 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. It is rare for more than one new version to be 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.

As the GDNQS is an important service, it must be made highly available which includes making it resistant to denial-of-service attacks. Countering denial-of-service attacks, such as flooding network connections to servers, is outside the scope of this article because they can only be dealt with at the network/operating system level. Other types of DoS attacks by outsiders are thwarted by the strong authentication of clients which is required for obtaining object creation tickets. Lacking credentials, outsiders will not be able to request any higher-level services from the GDNQS. Outsiders could only attack the authentication protocol itself. Hence, we assume a DoS-resistant authentication protocol is used [Dean and Stubblefield 2001].

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 it denies allocation requests if a producer has already been allocated his fair share. This policy should work fairly well; an analysis of the disk space requirements of all 16,187 SourceForge projects in October 2002 shows that only 10.2% of the projects used more than the average of 11.5MB. For large projects, AGOs could hand out special trace certificates (i.e., the certificates used to identify producers) that allow for more disk space usage at object servers. 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. This 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 frustrated 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, noncore 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 3.3.2) to aid with server selection.

5. 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 that a system will be 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.

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].

5.1 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 GDN, 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.

5.1.1 *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 marshaled 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). 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 in the following.

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 the 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 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]. This checkpointing scheme is implemented in our Globe prototype, see Bakker et al. [2003] for some preliminary experiments.

5.1.2 *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. Replica failures, if not reported by the replica's operating system, can in this case be detected using response times. In general, it is unclear how to accurately and securely detect the failure of components on the Internet. There is evidence to suggest that failure detection is practically feasible with a sufficient degree of reliability [Stelling et al. 1998]. Other replicas are found by querying the Globe Location Service. 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.

5.2 Failure Semantics

To remain manageable, GDN 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 [Cristian 1991]. Providing such semantics for downloads is easy since downloads in 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, 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. A failed upload will impact only a single DSO, not the whole

software collection. In addition, the number of uploads into a GDN DSO is low, reducing the number of actual failures. 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). For a complete discussion about the granularity of objects (e.g., how much software should be placed in a single object) and the details of this ad-hoc solution, we refer the reader to Bakker [2002].

We have 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].

6. RELATED WORK

We describe three related systems: Netlib [Dongarra 2004], SourceForge [Open Source Development Network 2004a] and Debian [Software in the Public Interest, Inc. 2004b].

Netlib is one of the earliest distribution networks of free software, in particular for scientific computing applications. Software can be retrieved using a variety of methods (gopher, email, FTP, and HTTP) from a set of mirror sites. The mirror sites are kept consistent using custom replication software. To enable the verification of software downloaded from a mirror, there is a PGP-signed index file containing the MD5 checksums of all published files. Fail-over to other mirrors is a manual procedure. Designed for the early and safer Internet, code would be uploaded as a single text file, and "the managing editors and, in some cases, an area editor will take at least a quick look to be sure the code seems suitable for netlib." [Grosse and Dongarra 1995].

At present, one of the primary sites for free software distribution is SourceForge. People submit requests for their projects to be hosted on SourceForge. When granted, they get access to Web and CVS servers and to a replicated file distribution service. The distribution service has a master/slave architecture. Files uploaded to the master server are replicated to a number of slaves around the world via the rsync protocol [Tridgell 2000] at 1 hour intervals. To download software, a user will normally look up the latest release on the project's Web page hosted by SourceForge. Selecting the release for download redirects the user to the distribution service. The service causes the user's browser to automatically initiate a download from one of the replicas, generally the one specified as the preferred replica by the user in an earlier download session. In case of replica failure, the user has to manually select another server to try. In the GDN, replica selection and fail-over is all automatic, and replicas are strongly consistent. The latter may help during flash crowds as all replicas can deliver the content from the moment of publication, not just the master server.

According to the SourceForge's usage terms and conditions, they do "not prescreen or review content". Instead SourceForge "reserves the right to refuse or delete any Content of which it becomes aware that it reasonably deems

not to fulfill the Purpose[. . .] [or] that it reasonably considers to violate the Terms or be otherwise illegal." [Open Source Development Network 2004b]. In sum, illegal distribution via SourceForge is combated by the application procedure for project space and an unknown procedure for detecting illegal content. The fact that all mirrors have the same content makes detecting a little easier since only one server has to be checked and cleaned. In the GDN, we do not assume that each replica has sufficient capacity to host the complete collection. This enables more organizations and private individuals to contribute resources. Content traceability enables us to remove illegal content from such a diverse hosting infrastructure.

An interesting project that prevents illegal distribution via their system is the Debian Project. Debian aims to provide a free operating system based on the Linux kernel. The system comes with over 8500 easy-to-install software packages maintained by over 800 people and replicated across more than 300 machines worldwide [Software in the Public Interest, Inc. 2004b]. To ensure the quality of the system and to prevent abuse, the publication of software is tightly controlled. People wishing to publish their software via the official Debian distribution system, which makes it conveniently available to Debian end users, have two options. The easy option is to find a so called sponsor among the ranks of the 800+ official Debian Developers. The sponsor will help the publisher to package his software following the Debian standards and will upload it to the distribution system on the producer's behalf. Although the software is listed under the producer's name, it remains traceable to the sponsor and is considered his responsibility. Illegal distribution is largely prevented as the sponsor will spot the most obvious attempts.

The difficult option is to become an official Debian Developer. Becoming a Debian Developer is a five-step process. First, a candidate must have his OpenPGP public key signed by an existing developer to establish the candidate's identity. To this extent, he has to preferably meet one of the developers in person and provide "a passport, a driver's license or some other ID" [Software in the Public Interest, Inc. 2004a]. If this is not possible, there are alternative ways such as sending a PGP-signed photo ID. Second, the candidate has to find a Debian Developer who is willing to act as the advocate for the candidacy. The advocate has to acknowledge that the candidate is ready to be a developer, that is, has the required skills and has been involved with Debian for some time. In the third step, the candidate has to pass an exam on the philosophy and procedures of the Debian Project. In the fourth step, he has to demonstrate his practical skills in, for example, package management or writing documentation. Finally, Debian's New Maintainer Committee will evaluate the results of the exams and grant or reject the request. Rejected candidates can generally reapply for developer status after a certain period.

Because becoming a developer is so difficult, only truly dedicated people who have invested considerably in Debian will apply for access. The chances of developers misbehaving are therefore expected to be small. The Debian procedure is considered a reputation-based scheme. For GDN's purposes, establishing the identity of the candidate is expected to be sufficient, and no extra testing of motivation is required. Cease-and-desist is thought to efficiently prevent illegal

distribution as long as the optimism is warranted and most producers are not malicious.

It is encouraging for the Globe Distribution Network that more than 800 people were willing to go through Debian's five-step application process. It shows that software producers will go through considerable effort to participate in a convenient system and suggests people may also be willing to do the formal identification step required for the GDN. Another important observation is that developers as sponsors are actually willing to act on behalf of others. This supports our idea for reporting illegal content via an intermediary in the case where the GDN has to be self-regulatory regarding illegal distribution.

Debian uses a custom replication system based on a master FTP server that pushes new releases out to the first tier of official mirrors once a day. Second-tier servers remain consistent with an official mirror via the pull-based rsync protocol. First-tier servers must mirror the complete master site. Second-tier servers are allowed to mirror just parts, allowing more people to participate in the distribution. The advantage of Debian is that the distribution system ties into the local package management system. End users can configure the system that manages the installation and deinstallation of software on their machine to use multiple servers, including a geographically close one by default. The management system can handle partial, inconsistent, and unavailable mirrors. As such, the Debian distribution system does not suffer the problems of SourceForge. Such a link to local package management is also possible with GDN, which also has no mirror inconsistencies nor does it requires manual fail-over, although this will require coordination among software producers regarding software packaging.

7. 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 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. Our research into efficient wide-area replication is continued in the Globule project [Pierre et al. 2002]. The GDN guarantees its availability despite attacks by outsiders 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 unmediated 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 shortly 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 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. Comm. *ACM* (Special Section on Adaptive Middleware) *45*, 6 (June).

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 of the USENIX Annual Technical Conference (FREENIX track)*. San Diego, CA. 141–152.

BAKKER, A., VAN STEEN, M., AND TANENBAUM, A. 2001. A law-abiding peer-to-peer network for free-software distribution. In *Proceedings of the IEEE Symposium on Network Computing and Applications (NCA'01)*. Cambridge, MA, IEEE Computer Society, 60–67.

BAKKER, A., VAN STEEN, M., AND TANENBAUM, A. 2003. A wide-area distribution network for free software. Tech. rep. IR-CS-002, Department of Computer Science, Vrije Universiteit Amsterdam. The Netherlands.

BAKKER, A., VAN STEEN, M., TANENBAUM, A., AND VERKAIK, P. 2003. Design and implementation of the Globe middleware. Tech. rep. IR-CS-003, Department of Computer Science, Vrije Universiteit Amsterdam, The Netherlands.

BALLINTIJN, G., VAN STEEN, M., AND TANENBAUM, A. 1999. Simple crash recovery in a wide-area location service. In *Proceedings of the 12th International Conference on Parallel and Distributed Computing Systems*. Fort Lauderdale, FL. 87–93.

BALLINTIJN, G., VAN STEEN, M., AND TANENBAUM, A. 2001. Scalable user-friendly resource names. *IEEE Internet Comput. 5*, 5 (Sept.), 20–27.

BURK, D. 2001. Copyrightable functions and patentable speech. *Comm. ACM 44*, 2 (Feb.), 69–75.

CARMODY, P. 2002. The world's first illegal prime number? `http://www.utm.edu/research/primes/curios/485...443.html`.

CHEN, P., LEE, E., GIBSON, G., KATZ, R., AND PATTERSON, D. 1994. RAID: High-performance, reliable secondary storage. *ACM Comput. Surv. 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 of the 11th International World Wide Web Conference*. Honolulu, HI.

CRISPO, B., POPESCU, B., AND TANENBAUM, A. 2004. Symmetric key authentication services revisited. In *Proceedings of the 9th Australasian Conference on Information Security and Privacy (ACISP'04)*. Sydney, Australia.

CRISTIAN, F. 1991. Understanding fault-tolerant distributed systems. *Comm. ACM 34*, 2 (Feb.), 56–78.

DEAN, D. AND STUBBLEFIELD, A. 2001. Using client puzzles to protect TLS. In *Proceedings of the 10th USENIX Security Symposium*. Washington, DC.

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

DONGARRA, J. 2004. The Netlib. `http://www.netlib.org/`.

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

GROSSE, E. AND DONGARRA, J. 1995. Subject: Notes to Netlib contributors. `ftp://ftp.netlib.org/misc/contrib`.

KATZENBEISSER, S. AND PETITCOLAS, F., EDS. 1999. *Information Hiding Techniques for Steganography and Digital Watermarking*. Artech House Publishers, Norwood, MA.

KUZ, I., VAN STEEN, M., AND SIPS, H. 2002. The Globe infrastructure directory service. *Comput. Comm. 25*, 9 (June). Elsevier Science, Amsterdam, The Netherlands. 835–845.

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

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.

OPEN SOURCE DEVELOPMENT NETWORK. 2004a. SourceForge.net open source software development Web site. `http://www.sf.net/`.

OPEN SOURCE DEVELOPMENT NETWORK. 2004b. SourceForge.net terms and conditions of use. `http://sourceforge.net/docman/display_doc.php?docid=6048&group_id=1`.

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

PIERRE, G., VAN STEEN, M., AND TANENBAUM, A. 2002. Dynamically selecting optimal distribution strategies for Web documents. *IEEE Trans. Comput. 51*, 6 (June), 637–651.

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

POPESCU, B., CRISPO, B., AND TANENBAUM, A. 2003a. A certificate revocation scheme for a large-scale highly replicated distributed system. In *Proceedings of the 8th IEEE International Symposium on Computers and Communications (ISCC'03)*. Kemer-Antalya, Turkey, 225–232.

POPESCU, B., CRISPO, B., TANENBAUM, A., AND ZEEMAN, M. 2003b. Expressing security policies for distributed objects applications. In *Proceedings of the 11th Cambridge International Workshop on Security Protocols*. Cambridge, U.K.

POPESCU, B., VAN STEEN, M., AND TANENBAUM, A. 2002. A security architecture for object-based distributed systems. In *Proceedings of the 18th Annual Computer Security Applications Conference*. Las Vegas, NV. 161–171.

SALTZER, J., REED, D., AND CLARK, D. 1984. End-to-end arguments in system design. *ACM Trans. Comput. Syst. 2*, 4 (Nov.), 277–288.

SANDHU, R., COYNE, E., FEINSTEIN, H., AND YOUMAN, C. 1996. Role-based access control models. *IEEE Comput. 29*, 2 (Feb.), 38–47.

SIMONS, B. 2000. From the President: To DVD or not to DVD. *Comm. ACM 43*, 5 (May), 31–32.

SIVASUBRAMANIAN, S., PIERRE, G., AND VAN STEEN, M. 2003. A case for dynamic selection of replication and caching strategies. In *Proceedings of the 8th International Workshop on Web Content Caching and Distribution (WCW'03)*. Hawthorne, NY.

SOFTWARE IN THE PUBLIC INTEREST, INC. 2004a. Debian GNU/Linux—Step 2: Identification. `http://www.debian.org/devel/join/nm-step2`.

SOFTWARE IN THE PUBLIC INTEREST, INC. 2004b. Debian GNU/Linux—The Universal Operating System. `http://www.debian.org/`.

STELLING, P., FOSTER, I., KESSELMAN, C., LEE, C., AND VON LASZEWSKI, G. 1998. A fault detection service for wide area distributed computations. In *Proceedings of the 7th IEEE Symposium on High Performance Distributed Computing*. Chicago, IL. 268–278.

TRIDGELL, A. 2000. Efficient algorithms for sorting and synchronization. Ph.D. thesis, Australian National University, Canberra, Australia.

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

VAN STEEN, M., HAUCK, F., AND TANENBAUM, A. 1998. Locating objects in wide-area systems. *IEEE Communications*, 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/en/ip/wct/`.

Received February 2003; revised September 2003 and July 2004; accepted March 2005