Global Distribution of Free Software (and other things)

Arno Bakker, Ihor Kuz, Maarten van Steen,
Andrew S. Tanenbaum, Patrick Verkaik
Vrije Universiteit Amsterdam
{arno,ikuz,steen,ast,patrick}@cs.vu.nl

Abstract

The Globe Distribution Network (GDN) is a distributed system designed to support the secure distribution of free software. Software packages are encapsulated into distributed objects that implement their own strategy for replicating state. This approach allows each package to be replicated in a way that best handles client demands or optimizes usage of network resources. There is no single global strategy by which all packages are distributed, as is actually the case when applying existing techniques such as for mirroring FTP sites. GDN is integrated into the Web making it possible for clients to use any existing Web browser for downloading a package from the nearest client. In addition, GDN provides mechanisms against illegal distribution of content by tracing content to its publisher. Publishers may be protected by letting them use a pseudonym.

1 Introduction

The Internet facilitates myriad services for distributing information. One of the more popular services is the distribution of software, exemplified by CNET, Tucows, Freeware, and many others. The traditional approach for distributing software (as well as other files), is to set up a server that gives read-only access to a collection of files by means of protocols such as FTP or HTTP. Although this approach in general works reasonably well, it potentially introduces two important, related problems.

First, because there is only a single access point the availability of the service is determined completely by that of the server. Second, this server can easily become a performance bottleneck (which, in turn, affects the availability). Not only may the server need to provide simultaneous access to many users, it should also be capable of providing high throughput. Performance may further be affected by the network connection between client and server. Especially when dealing with long-haul connections, there are restrictions on available bandwidth that fully determine the time to access and download files.

The traditional solution is to simply replicate the collection of files to a *mirror site*. Mirroring can dramatically improve availability and performance, especially if brute-force techniques are used to guarantee high throughput to nearby clients. For example, recently NLUUG's software archive (forming a mirror for many large collections of files and documents) has been linked to a high-speed national network through a 1-Gbit interface. This connection now provides excellent access to local (i.e., national) clients.

Mirroring introduces a consistency problem: a file copy needs to be synchronized with its master copy or otherwise a client may be downloading an old version of the file instead of the most recent one. Consistency problems can be partly solved using tools such as *rsync*, but leaves other issues open. For example, in many cases it is unclear whether it actually makes sense to copy *all* files to a mirror site. If it is required to efficiently use local resources, such as disk space, it may be better to mirror only the popular files. Likewise, deciding on when and how to synchronize copies reduces to being an educated guess if no information on usage and access patterns is available. Finally, mirroring is not transparent to the user. When looking for a file, the user will have to select the mirror site to go to.

Mirroring, and other alternative solutions such as adopted by Content Delivery Networks and peer-to-peer systems, still leave a number of important issues open. For example, a file-distribution system should be able to provide security, simple facilities for uploading files, dynamic replication to where files are mostly needed (e.g., to handle flash crowds), different protocols for maintaining consistency between file copies, and should be able to span multiple administrative organizations.

The *Globe Distribution Network* (GDN) is set up to partially solve these problems [2]. It is developed as an application of the Globe wide-area distributed system [31]. GDN consists of a collection of servers that store packages of files. It has the following important properties:

- Clients are transparently directed to the nearest available copy of a package
- It can be used with existing client-side software (i.e., Web browsers)
- It has facilities for preventing illegal content distribution
- A package can be replicated and migrated in a package-specific way

GDN is currently running in an experimental setup worldwide with servers in The Netherlands, France, United States, Brazil, and Israel. Besides software distribution, it is also hosting Web content. The source codes and binaries (mostly Java bytecodes) for various platforms are freely available (of course, through GDN) and can be accessed at http://www.cs.vu.nl/globe.

In this paper we provide an overview of GDN. Our main contribution is that we provide a description from the perspective of someone wanting to set up and maintain a GDN site. This description excludes many specific details on the GDN internals, but instead opens the road for participation in GDN. We start by discussing related work in Section 2, followed by an architectural overview in Section 3. A description of the components that comprise a GDN site is given in Section 4. Security is described in Section 5. We provide an outlook and conclude in Section 6.

2 Related Work

There are a number of systems that address replication of files. In the following, we briefly consider distributed file systems, content delivery networks, and peer-to-peer storage systems.

2.1 Distributed File Systems

Distributed file systems have traditionally been weak in providing sophisticated support for file distribution and replication, mainly because they have not been designed to operate across wide-area systems. A notable exception is AFS, which has gradually evolved to a wide-area system in which scale is obtained through client-side file caching in combination with callbacks [27]. A similar approach has recently been adopted for NFS, by which its traditional stateless-server design is abandoned [18, 26]. In these approaches, when a file is opened it is transferred to the client where all operations can now be performed locally. Updates are propagated back to the servers when the file is closed. Server replication is primarily supported only for increasing availability. In all cases, the same strong consistency model is implemented for all files; there is no differentiation between files.

Weaker consistency models have also been adopted. For example, in Ficus, each workstation effectively acts as a server and potential replica host for the files that a client wants to access [17]. Updates are propagated in a lazy fashion using anti-entropy [7], but because they can be initiated concurrently at different sites, there is a need for reconciliation when write-write conflicts occur. Likewise, Coda supports disconnected operation allowing a client to continue updating its local copy of a file, without any guarantees that another client may also be performing updates [11]. More important is that in these examples replication of files is again hard-coded into the system. At best, they allow a callback to user-provided component for conflict resolution.

2.2 Content Delivery Networks

More recently, content distribution networks (CDNs) are providing services that adopt a more flexible approach to dynamically distributing and replicating Web content. In RaDaR, files are dynamically replicated and migrated between a collection of servers depending on the access statistics of clients [22]. Akamai uses a proprietary algorithm to decide where to place replicas, but uses a relatively simple cache-invalidation scheme to refresh copies on demand [13].

An important contribution of CDNs is that they address placement issues. Unlike many other distributed storage systems, CDNs attempt to place replicas of a Web document only where it is really needed. Various studies on the optimal placement of Web content have been conducted (see, e.g., [20]). One of the issues that makes optimal placement such a difficult problem is that it is not trivial to select a performance metric that leads to significant differentiation between placement decisions [15]. In other words, it is hard to evaluate what makes one placement strategy better than the other. It may be that combining geographical distance at the level of continents with distance expressed as hop counts between autonomous systems at lower levels is currently the best we can do [21].

Current CDNs and related Web replication schemes generally use a single protocol for keeping copies consistent. Only relatively few approaches have been studied in which the protocol itself is adaptive, for example, by allowing to dynamically switch between pushing and pulling updates [8, 32]. Our own studies regarding replicating Web content indicate that adaptive protocols may help to simultaneously optimize bandwidth usage and reduce client-perceived latency [19]. It is yet unclear whether such adaptivity is also necessary for the distribution of software packages.

2.3 Peer-to-Peer Systems

An important emerging area of related research is in file-sharing peer-to-peer networks. In the novel approaches, these networks consist of two separate layers. The lowest layer consists of (probabilistic) routing and searching protocols, which essentially allow a client to route a request based on a unique key from a large number space. This approach to routing is adopted, for example, in Tapestry [33], Chord [28], and Pastry [24].

Storage facilities are provided by a separate layer. Invariably, a client is offered a massively scalable system that supports immutable files [6, 23, 25]. Updates are handled through versioning. Security is often provided through data encryption and special measures to protect against denial of service attacks. However, none of these systems offers a means to deal with illegal content distribution, which is one of the goals of GDN.

Another potential shortcoming of the current peer-to-peer networks that needs further investigation, is the way that network proximity is taken into account. Some systems, such as Freenet [4] simply ignore network proximity when routing requests. Other systems give higher priority to nodes that are close by in terms of round-trip delays. However, it is unclear whether this metric is actually sufficient to achieve efficiency in terms of network-resource usage.

3 GDN Architecture

Let us now concentrate on the overall organization of GDN. We first consider its main components that together form the core of the distribution network. GDN by itself is constructed as an independent distributed system built on top of the Internet's transport layer. To integrate it into the Web, thus allowing existing clients to interact with GDN, special measures need to be taken. These measures are discussed separately.

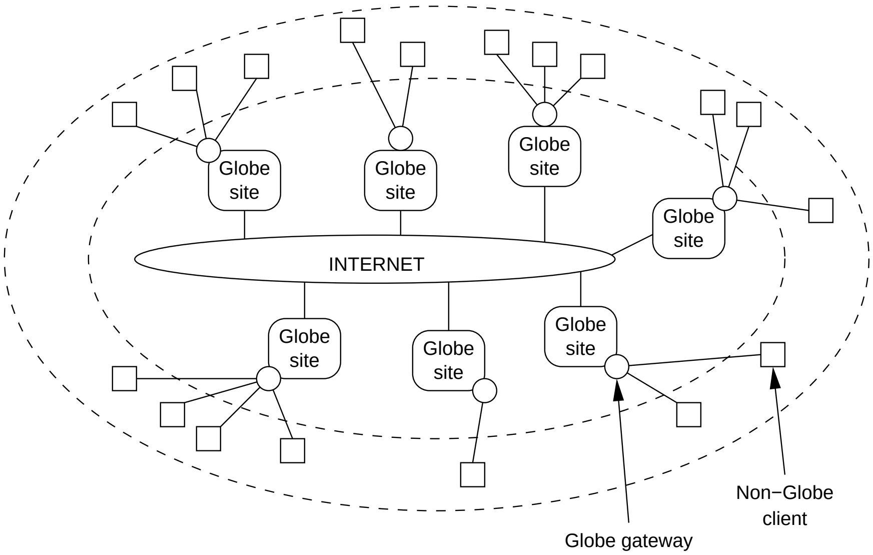

Figure 1: Concentric layering of GDN.

3.1 GDN Core Network

GDN has been constructed as an application of Globe, a wide-area distributed system developed at the Vrije Universiteit Amsterdam [31]. One of the key concepts underlying Globe is that shared data is encapsulated by distributed objects. Unlike most other object models, objects in Globe can be physically distributed and replicated across multiple machines. Important is that each object not only encapsulates its state and the implementation of operations on that state, but also that it implements its own distribution strategy. In other words, each object separately implements a strategy that governs how its state is partitioned, replicated, and migrated between hosts. Likewise, each object carries it own implementation for security, persistence, and so on.

Each Globe system is normally set up as a collection of *object servers* for hosting distributed objects. GDN is no exception and also consists of a number of Globe object servers (GOSs). Conceptually, an instance of GDN consists of two concentric layers, as shown in Figure 1. The inner layer is formed by a collection of *sites* each running Globe servers and Globe-enabled clients. The outer layer consists of standard Internet and Web clients that access GDN services by means of gateways. In this section, we consider only the inner layer.

The Globe object servers comprising GDN collaborate in storing, distributing, and replicating files. Files that are to be stored in GDN are encapsulated in objects that are physically distributed across multiple Globe sites. We refer to these objects as *(distributed) package objects*. A package object typically represents a version of a software package (also known as a *revision* [5]), including its variants for different file formats. Different versions of the same software package are contained in separate package objects. In this way, we obtain the flexibility to adjust the distribution of a software package to its popularity. In this way, old versions may be stored at only a few places, whereas the most recent version of a package is replicated across many sites, for example, in anticipation of flash crowds, or simply for high availability and performance. Again, note that GDN allows the distribution of its package objects to be tuned per object.

A package object consists of a number of *local representatives*, one on each server on which the object is hosted. A local representative thus contains the files that comprise a software package. A client, however, need not know what a local representative actually contains. Such details are hidden behind an object's interfaces, which are the only thing a client gets to see. A local representative, in turn, is implemented as a local (i.e., nondistributed) object. Such a local object is hosted by an object server similar to the way that local objects in systems such as CORBA and Legion are hosted. We return to these issues below.

Each package object has an associated human-friendly global name similar to a UNIX pathname. An example of such a name is /nl/vu/cs/globe/proj/home. Names in GDN are globally defined. In other words, all package names together form a worldwide global name space. To access a package in GDN, a client will pass a name for the package to the GDN naming service, which will eventually return the address of on object server hosting a replica of the package. The naming service aims to return the address of an object server that is close to the client. To this end, we have split the naming service into two parts.

The first part supports human-friendly names that are bound to identifiers known as *object handles*. An object handle is a stable and location-independent reference to package object, comparable to a UUID as used in DCE and now available on many (distributed) computer systems such as Linux. As we explain in detail later, maintaining name-to-object bindings is implemented in GDN by means of DNS.

The second part of the naming service is formed by a separate and specially developed Globe location service, which is responsible for maintaining the bindings between an object handle and the current addresses of the replicas of a package. By separating human-friendly names from addresses, we have been able to develop a worldwide scalable service that allows efficient lookups of addresses even if addresses change regularly [3].

The location service returns a contact address describing exactly where and how the requested package object can be contacted. A contact address usually contains the address of an object server along with a server-specific local-representative identifier. Also, the contact address contains an implementation identifier, pointing, for example, to a Java jar file that the client should load in order to set up a connection with the object. In a sense, the GDN contact addresses are similar to the interoperable object references (IORs) used in CORBA [16]. Different types of contact addresses are supported, but these differences are all hidden to GDN clients. We return to these issues when discussing object servers.

3.2 Integration in the Web

Many users on the Internet prefer to use their existing client-side software to access services such as offered by GDN. We did not find it reasonable to force users to install GDN-aware clients in order to be able to access GDN. Instead, we have chosen to provide a means that allows the use of standard browsers to download the files contained in package objects. However, at this moment, uploading files requires using one of our GDN clients. In the following, we briefly describe how GDN is integrated into the Web.

Naming Issues

A minimal requirement for integration in the Web is that we adopt a naming convention that is accepted by Web browsers. In practice, this means that a user should be able to use a browser-recognizable Uniform Resource Identifier (URI). Unfortunately, not all browsers are capable of recognizing the same URIs. In some cases, such as Netscape, the types of URIs that are supported is fixed, Other browsers, like the Mozilla version of Netscape and Internet Explorer provide more flexibility.

We have decided to take a two-step approach. First, Globe pathnames are extended to *Globe URNs* by attaching the prefix “globe:/” to each pathname, leading to names such as globe://nl/vu/cs/globe/proj/home. Underlying this naming convention is that extensible browsers should be able to easily support new scheme identifiers such as “globe.”

However, most browsers recognize only pre-configured scheme identifiers. We therefore take a second step by embedding Globe names into *embedded Globe URNs*, which are plain HTTP URLs. A name such as http://globe.cs.vu.nl:23003 is added as a prefix to a Globe name, resulting in, for example, the embedded Globe URN http://globe.cs.vu.nl:23003/nl/vu/cs/globe/proj/home. This URL refers to an object known under the Globe name /nl/vu/cs/globe/proj/home. We allow a user to specify any prefix he likes, but every prefix requires the support from specially configured Globe HTTP servers. As we explain next, these servers are responsible for handling Globe URNs.

Globe HTTP server

To support downloading of files through Web browsers, we make use of a sitewide service that translates HTTP requests into read operations on the appropriate package object and returns the results of an operation as an HTTP reply. This service, referred to as the *Globe HTTP server*, consists of two parts, each implemented as a separate process as shown in Figure 2.

One part is called the *Globe gateway* and acts as a Globe client that can *bind* to any package object. When a process binds to a package object a local representative of that object is loaded into the process’s address space. The Globe gateway accepts HTTP requests that contain a Globe URN referring to the requested package by its name as explained above. Note that the gateway uses HTTP only as

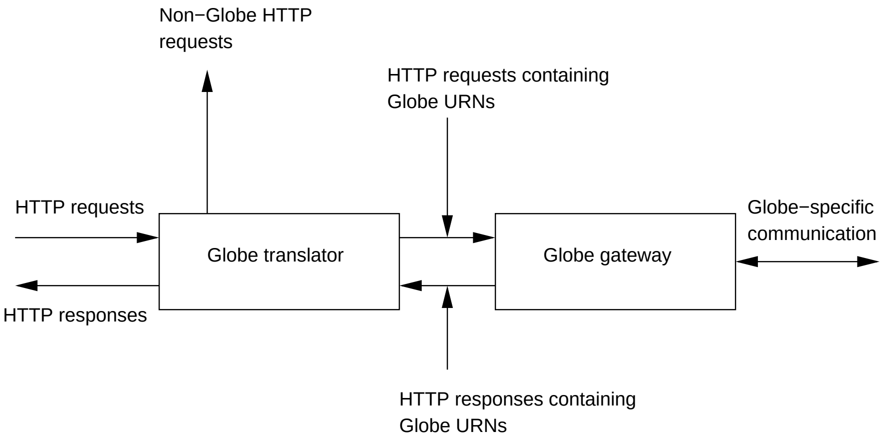
Non−Globe HTTP requests
HTTP requests containing Globe URNs
HTTP requests
Globe translator
Globe gateway
Globe−specific communication
HTTP responses
HTTP responses containing Globe URNs

Figure 2: The Globe HTTP server.

its communication protocol; the names it accepts are always expressed as Globe URNs (i.e., with URI scheme identifier globe). The gateway then binds to the named package object, and executes a read operation to download a file from the package.

The other part of a Globe HTTP server is the *Globe translator*. This part acts as a regular HTTP proxy server that accepts incoming HTTP requests containing URLs having http as their scheme identifier. Each translator is configured to recognize a specific prefix as explained above. A URL containing this prefix is treated as an embedded Globe URN and is translated into a Globe URN. This URN is then passed to the Globe gateway. The translator also accepts HTML pages that have are returned by the gateway. Any Globe URNs contained in such a page is translated into its embedded, after which the modified page is passed on to the client.

In order for a client to access the nearest Globe HTTP server, we apply a redirection mechanism that is based on the NetGeo service from CAIDA (see `http://www.caida.org/tools/utilities/netgeo/` for more information). NetGeo maintains a mapping between IP addresses and geographical location. When a non-Globe client accesses GDN, it does so by sending an HTTP request to a server at a fixed location (in our case, the location with domain name enter.globeworld.org, but we can support other names as well). Using NetGeo, we compute the Globe HTTP server that is closest to the requesting client, and return an HTTP response requesting the client to refresh the returned HTML page at the selected Globe HTTP server. In effect, we thus establish an HTTP redirect.

4 GDN Internal Organization

Internally, each Globe site consists of a number of servers that jointly comprise the implementation of Globe for that site. We distinguish four different types of servers: an object server for hosting the local representatives of distributed (package) objects, a name server to implement the Globe naming service, a location server for implementing the location service, and finally a directory service to provide sitewide information. The organization of these servers and role with respect to the respective Globe services are discussed in the following.

4.1 Object Server

A crucial server in GDN is the *Globe Object Server* (GOS). A GOS is responsible for hosting a replica of a package object. Different package objects may be hosted by the same GOS. It is also possible to run several object servers at the same site.

Replication Support

An important aspect of the object server is that it provides the facilities for replicating an object. In principle, for each hosted package object a GOS offers a *contact point* that allows other processes to contact the object. A contact point is described by means of a *contact address*, which contains exact information on how and where to reach an object. A contact address is generally made publicly available (e.g., by means of the Globe location service) to allow clients to *bind* to the object, as we explained above.

Globe supports different types of contact addresses. A typical example is a contact address that contains the IP address of the object server, a TCP port the server is listening to, and an index identifying the local representative of the package object. In this case, a process binding to the object may be required to install only a standard client-side stub that effectively establishes RPCs over TCP/IP. A completely different example is a contact address consisting of a Java object reference, in which case a client will only have to deserialize (i.e., unmarshal) the reference in order to contact the object. We have used this latter mechanism to wrap a CORBA-like system around Globe [9].

The first copy of a package object is always explicitly created on an object server using our GDN client. Besides offering processes the facilities to bind to a hosted package object, a GOS can also be requested to create a replica. A replica of an existing package object is normally created by instructing an object server to bind to that object using a contact address that specifies the type of replica that should be created. The effect is that the binding object server installs a local representative for that object and that the files contained in the package are transferred to the new replica.

It is important to note that the object server has no hard-coded policy concerning the way its hosted package objects are replicated. Each object carries its own implementation of such a policy; the only thing the object server does is ensure that an object's policy is carried out. For example, consider an object server that is hosting the local representative of a package object that is replicated according to a master/slave strategy. If the local representative is the master, then it will contain code by which updates are eventually propagated to its slaves (which are hosted by other object servers). The package object decides when and how update propagation takes place; the object server simply provides the means to allow communication with other servers.

Persistent-object support

Packages are implemented as persistent objects. A persistent object in Globe is defined as an object that can continue to exist even it is not currently hosted by an object server. In contrast, a transient (distributed) object can exist only if there is at least one server that is hosting the object. To implement a persistent object, an object server needs to ensure that the state of the local representative it is hosting is written to persistent storage when the server is shut down. Likewise, that state should be able to be read from storage by another server at startup time.

To facilitate persistence, our object servers allow contact points to become persistent as well. Persistent contact points remain valid while (the local representative of) an object is not running; when the object is running again, its contact points can be used as before.

Fault tolerance

Related to persistence is the support for fault-tolerant objects. At present, GDN provides only minimal fault tolerance by periodically storing the complete in-memory state of a local resprentative to disk. If the server crashes during operation, the most recently saved state is restored effectively recovering local representatives. No checkpoint is made if the state has not been altered since the last checkpoint.

We decided to implement just periodic checkpointing for performance reasons. The alternative is to checkpoint the state at each update operation, but this was felt too expensive as it requires a synchronous disk operation. Further research is needed to see whether we can improve this situation. In particular, we intend to explore the promising combination of object-specific fault tolerance and replication for performance. Some initial work on this matter is described in [10].

When recovering from a server crash, the object server normally contacts the master replica to see if there have been updates. This synchronization is necessary to ensure consistency. Before doing so, the server first checks whether any updates have occurred during the crash period to avoid needless state transfer across the network.

4.2 Naming Service

As mentioned in the previous section, each package object can have a human-friendly name that is resolved to an object handle. To this end, the Globe name space consists of a strict hierarchical name space that is represented as an edge-labeled rooted tree with each node representing either a directory or a package. This name space is implemented using DNS technology. For example, an object name such as /nl/vu/cs/globe/proj/home is internally rewritten to the DNS name home.proj.globe.cs.vu.nl. and passed to a local DNS server to be resolved.

A node in the Globe name space is implemented as a DNS TXT record. In this way, we can use existing implementations of DNS to implement our name space. It is important to note that we do not require DNS to be replaced. Instead, we are making use of the existing DNS servers and organization, and expand the current DNS name space by adding leaf domains.

A directory is represented by a DNS domain name and has a special TXT record associated to it containing the keyword GLOBEDIR and the zone name in which that directory is supposedly contained. For example, the DNS entry

```
$ORIGIN org.
globeworld IN TXT "GLOBEDIR globe.cs.vu.nl."
```

identifies the directory with Globe name /org/globeworld, which is maintained by the DNS server at globe.cs.vu.nl. Directory entries can contain subdirectories or object handles, each of which are again implemented as TXT records. The following entries identify two subdirectories /org/globeworld/bin and /org/globeworld/proj, as well as an entry /org/globeworld/fozzie storing an object handle:

```
$ORIGIN globeworld.org.
bin    IN TXT "GLOBEDIR globe.cs.vu.nl."
proj   IN TXT "GLOBEDIR globe.cs.vu.nl."
fozzie IN TXT "GLOBEOBJHANDLE ACARZJgItYlPUkSIB590cUN42g/x=="
```

A Globe name server consists of two processes normally colocated on the same machine. One process is a DNS name server running BIND, which is actually responsible for storing and maintaining the name space. The other process is a naming authority, responsible for securely updating the local DNS database on behalf of clients. Updates are actually carried out by BIND, but which accepts update requests only from its naming authority.

Organizing the Globe name server as a colocated pair of processes is primarily done for simplicity. First, letting BIND accept updates from only one process avoids expensive locking schemes to handle nonatomic update operations. For example, removing a directory requires a read operation to first check whether the directory is empty, and a subsequent write operation to actually remove it. Second, if a naming authority is reponsible for only a single domain, then updates for only that domain are affected if the naming authority is disrupted. Finally, colocating the naming authority and the DNS name server has the advantage that the former

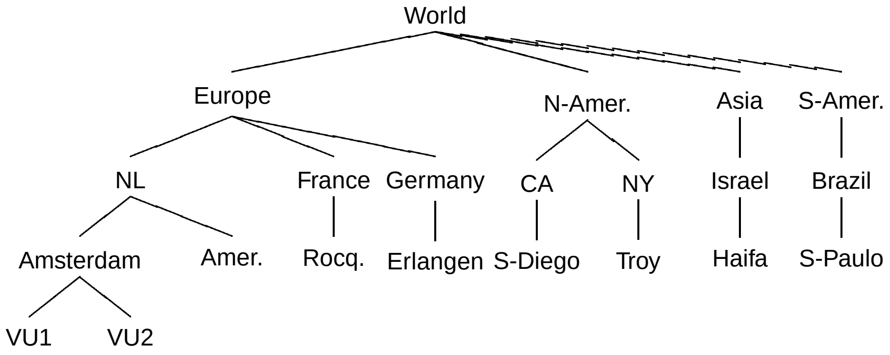

Figure 3: The current organization of the Globe location service.

can be identified using a normal DNS lookup request. The IP address returned by the DNS name server is the same one as its associated naming authority.

4.3 Location Service

The Globe Location Service (GLS) is designed to provide a worldwide scalable solution to locating mobile and replicated objects. For each object, GLS maintains a mapping between an object's object handle and the contact addresses where the object's replicas can be found. Only those replicas that can be used for binding a client to the object need to be registered with GLS.

The location service is organized as a worldwide distributed search tree, based on a hierarchical partitioning of the underlying network into domains. The top-level domain covers the entire network, whereas the lowest-level domains typically correspond to a moderately-sized network such as a university campus of the office network of a corporation's branch in a certain city. In the current setup, we have a relatively small five-level tree, as shown in Figure 3.

In this setup, each node is represented by a separate server. Each server maintains location information on the package objects that reside in its associated domain. This location information is either a contact address or a pointer to a lower-level domain where the object resides. Location information is stored in a *contact record*. For example, consider a package that has been replicated to the VU1 and the Erlangen domain. The server at Erlangen will store the object's contact address, whereas the server for Germany will store a pointer to the Erlangen server, and so on, leading to the situation shown in Figure 4.

To look up a contact address, a client contacts its nearest leaf node. If that node does not contain any location information on the requested package object, the request is forwarded upwards until a node is reached that does. In the worst case, a lookup request travels to the root. The root and intermediate nodes store only forwarding pointers to child nodes. Therefore, the lookup request subsequently travels a path of forwarding pointers down to one of the leaf nodes.

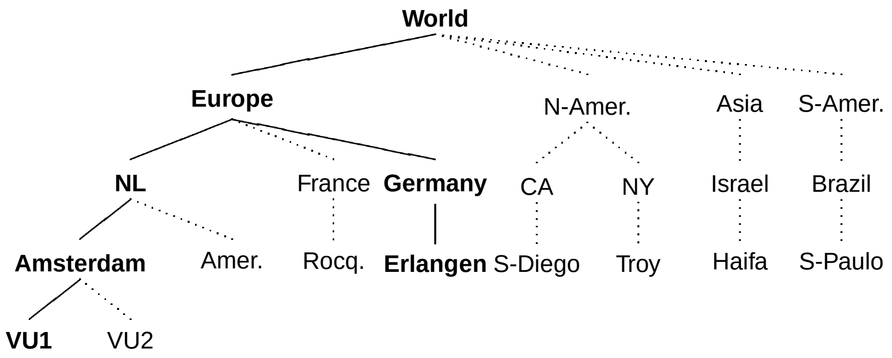

Figure 4: Nodes containing a forwarding pointer or address for a package replicated at VU1 and Erlangen.

Returning to our example tree of Figure 4, suppose a client in Rocquencourt (France) issues a lookup request. That request is first processed at node Rocquencourt, from where it is forwarded first to France and then to Europe. The latter stores two forwarding pointers: one to node NL and one to Germany. In such cases, an arbitrary choice is made in the current setup. Assuming that the request is forwarded to Germany, it eventually reaches node Erlangen where a contact address is found, which is then returned to the client.

An important observation is that the lookup request travels only between servers in the smallest domain in which both the client and the requested package reside. In a similar fashion, we exploit locality for update operations. Details on these operations, as well as various important refinements and optimizations are described in [30, 1].

Implementing GLS as a single-rooted tree obviously introduces a scalability problem for higher-level nodes. The solution to this problem is to apply partitioning techniques. In other words, we implement a logical node using multiple servers. It is beyond the scope of this paper to describe these techniques in any detail. However, it is worth noting that it is possible to distribute contact records in such a way that hosts running servers for GLS are equally loaded while maintaining the desirable locality properties of having a search tree. Details on such partitioning schemes, along with a description of simulation experiments are described in [29].

4.4 Infrastructure Service

Each site, finally, also runs an LDAP-based directory server that contains virtually all configuration information of a site, but can also provide information to remote clients on local resources. These directory servers jointly constitute a worldwide directory service known as the Globe Infrastructure Directory Service (GIDS).[1] A

[1] GIDS is Dutch for “guide.”

GIDS server can be located using its DNS name, which corresponds to the DNS name of the site the server is part of. In this sense, GIDS is similar to Active Directory [14], an important difference being that GIDS servers constitute only leaf nodes in the name space. Intermediate nodes are always DNS servers. This difference makes GIDS more efficient. Details on GIDS are described in [12].

5 Security

One of the more profound aspects of GDN is its support for security. Obviously, a system such as GDN should be able to provide protection against various security attacks. In addition, and perhaps more important considering the various peer-to-peer file-sharing mechanisms currently under siege, was whether GDN should also disallow publication of illegal content such as copyrighted material by means of content moderation. In that case, all content that would be uploaded to GDN would first need to be checked. We decided that this kind of protection was not what GDN should offer. Instead, GDN should primarily provide the means to *securely distribute* content. In addition, we decided to provide mechanisms that would allow publishers of illicit content to be banned from using GDN.

To this end, we support a security scheme that is based on traceable files. In this scheme, each person publishing content in GDN (a *GDN producer*) is required to digitally sign all the content he or she publishes, thereby making it traceable to that person. The GDN checks that uploaded content is traceable and refuses content that is not. The purpose of the digital signatures is to be able to trace content back to its original publisher such that when illegal content is found (copyrighted works, offensive material, malicious or forbidden software) this publisher's access to the GDN can be revoked, and his publications removed. Details on this scheme are described in [2].

When a software maintainer wants to start publishing his software through the GDN he has to contact one of the so-called *access-granting organizations* (AGOs). An AGO verifies the candidate's identity by checking his passport or other means of identification. In addition, the organization checks if this person has not been banned from the GDN by any of the other AGOs. If the candidate is clean, the AGO 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 AGO basically acts as a Certification Authority for GDN users.

Every owner of an object server specifies which producers it wants to give access to his object server. In principle this is done at AGO-granularity: the 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 create local representatives and place content on that owner's object

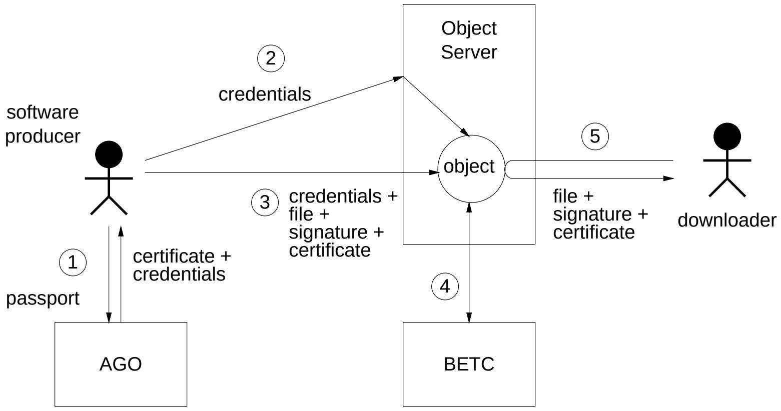
Object Server
2
credentials
software producer
object
5
3
credentials + file + signature + certificate
file + signature + certificate
downloader
1
certificate + credentials
passport
4
AGO
BETC

Figure 5: Basic GDN security model.

server.

An upload now proceeds as follows. Assume the producer has created a package 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 package object along with the file. When the upload is finished the object verifies the trace signature. If the signature is false, either because the producer has been banned from the GDN, the certificate did not contain the right public key, or the file did not match the digital signature, the object removes the uploaded file from its state. As only files that are provided by an active producer and that carry a valid signature are allowed in an object, all content in the GDN is always traceable.

In the current implementation the trace signatures are not checked by the object itself for technical reasons. Instead trace signatures are checked by an external service, called *BETC*, that runs in parallel to each GDN object server. The upload process is illustrated in Figure 5, and consists of the following five steps.

1. To get access to the GDN a software publisher identifies himself to an AGO and receives a trace certificate and Globe credentials in return.
2. The producer requests an object server, using his Globe credentials, to create a GDN package.
3. The producer creates a digital signature for the archive file and uploads it along with the file and the trace certificate into the GDN package.
4. BETC checks if the file and signature match, and has the object server destroy the replica of the object if not.
5. A user downloads the archive file, trace certificate and trace signature from the GDN object and verifies that they match.

For the current setup of GDN we are running a single AGO at the Vrije Universiteit, called Fuego. Fuego is primarily running for demonstration purposes, it is by no means to be considered as a certification authority such as, for example, Verisign. In order to participate in GDN, however, each new GDN producer will have to obtain a certificate from Fuego by sending an e-mail. This certificate will contain the maintainer's public key used to securely trace published material, and is signed by Fuego.

Details concerning the configuration of various security mechanisms, including the local storage of key pairs, can be found in the Globe Operations Guide that is part of the distribution of GDN.

6 Practical Experience and Outlook

GDN as it stands today is the result of combining scientific efforts and development work. Development has largely taken place in the form of an externally supported project called SIRS ("Scalable Internet Resource Service"). One of the original goals of SIRS/GDN was to come to a solution for offloading popular FTP sites. The research and development efforts put into GDN amount to approximately 12 person-years of work, excluding the work put into the Globe location service (which is at least another eight person-years).

GDN is now running on multiple sites across the Internet. There are various "local" sites hosted in The Netherlands; the main international sites are (in alphabetical order):

- Amerongen, The Netherlands (NLnet Foundation)
- Amsterdam, The Netherlands (Vrije Universiteit)
- Ithaca, New York (Cornell University)
- Haifa, Israel (Technion)
- Rocquencourt, France (INRIA)
- San Diego, California, USA (CAIDA)
- Sao Paulo, Brazil (University of Sao Paulo)

GDN is hosting software for Linux (2.4 kernels and the RedHat 7.2 updates), Amoeba, and Minix. We are also hosting various Web sites. Experiments have begun to host a large portion of the SourceForge database, replicating it across San Diego (CAIDA), Redwood City (Vixie Enterprises), and Amsterdam (VU). These experiments still need to be completed and require additional development work at the VU.

> *We are continuing our efforts to host content and welcome any suggestions. Also, parties that are willing to participate in setting up a larger GDN network are very welcome.*

Research and development on GDN continues. At present, our group is actively conducting research in the following areas:

Adaptive replication We are currently looking into dynamic replication of Web documents. So far, this research has provided us insight in how and when to evaluate access traces in order to decide what a best replication strategy is [19]. Future research will concentrate on the selection of the best location to place a replica. Also, we will take a look at dynamically switching the consistency protocol so as to optimize network resources and access delays. The results of these research will be embedded into future GDN releasees.

Systems management: We are also currently looking at management issues. In particular, we considering the problem how we can automatically bring up an entire, worldwide-spanning location service tree without initially having to physically distribute that tree across the Internet as well. This research affects the (semi-)automatic distribution of GDN across multiple sites. In addition, we have recently started to concentrate on more general systems management issues related to distributed systems such as GDN. For example, we are currently working on a general scheme to add and remove servers comprising a distributed service.

In conclusion, we feel that we have just made a start with GDN when it comes to its deployment. However, we are continuing the research that has been set out as part of GDN and expect to enhance the current system in terms of size, content, and functionality.

7 Acknowledgements

The development of the current version of GDN could not have been accomplished without the financial support from the NLnet Foundation. In addition, the discussions with Teus Hagen, Wytze van der Raay, and Frances Brazier concerning the deployment of GDN have significantly contributed to making GDN publicly available as soon as possible. Wytze van der Raay deserves special credit for being our first and highly constructive beta tester. We also gratefully acknowledge the contributions from Egon Amade and Ivo van der Wijk.

References

[1] A. Baggio, G. Ballintijn, M. van Steen, and A. Tanenbaum. “Efficient Tracking of Mobile Objects in Globe.” *Comp. J.*, 44(5):340–353, 2001.

[2] A. Bakker, M. van Steen, and A. Tanenbaum. “A Law-Abiding Peer-to-Peer Network for Free-Software Distribution.” In *Proc. Int’l Symp. Network Computing and Applications*, Cambridge, MA, Feb. 2002. IEEE.

[3] G. Ballintijn, M. van Steen, and A. Tanenbaum. “Scalable Human-Friendly Resource Names.” *IEEE Internet Comput.*, 5(5):20–27, Sept. 2001.

[4] I. Clarke, O. Sandberg, B. Wiley, and T. Hong. “Freenet: A Distributed Anonymous Information Storage Service.” In H. Federrath, (ed.), *Designing Privacy Enhancing*

Technologies, volume 2009 of *Lect. Notes Comp. Sc.*, pp. 46–66. Springer-Verlag, Berlin, 2001.

[5] R. Conradi and B. Westfechtel. “Version Models for Software Configuration Management.” *ACM Comput. Surv.*, 30(2):232–282, July 1998.

[6] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. “Wide-area cooperative storage with CFS.” In *Proc. 18th Symp. Operating System Principles*, Baniff, Canada, Oct. 2001. ACM.

[7] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. “Epidemic Algorithms for Replicated Data Management.” In *Proc. Sixth Symp. on Principles of Distributed Computing*, pp. 1–12, Vancouver, Aug. 1987. ACM.

[8] V. Duvvuri, P. Shenoy, and R. Tewari. “Adaptive Leases: A Strong Consistency Mechanism for the World Wide Web.” In *Proc. 19th INFOCOM Conf.*, pp. 834–843, Tel Aviv, Israel, Mar. 2000. IEEE.

[9] M. Jansen, E. Klaver, P. Verkaik, M. van Steen, and A. Tanenbaum. “Encapsulating Distribution in Remote Objects.” *Information and Software Technology*, 43(6):353–363, May 2001.

[10] J. Ketema. “Fault-Tolerant Master-Slave Replication and Recovery in Globe.” Master’s thesis, Vrije Universiteit Amsterdam, Mar. 2001.

[11] J. Kistler. *Disconnected Operation in a Distributed File System*, volume 1002 of *Lect. Notes Comp. Sc.* Springer-Verlag, Berlin, 1996.

[12] I. Kuz, M. van Steen, and H. J. Sips. “The Globe Infrastructure Directory Service.” *Comp. Comm.*, 25(9):835–845, June 2002.

[13] F. Leighton and D. Lewin. “Global Hosting System.” United States Patent, Number 6,108,703, Aug. 2000.

[14] A. Lowe-Norris. *Windows 2000 Active Directory*. O’Reilly & Associates, Sebastopol, CA, 2000.

[15] K. Obraczka and F. Silva. “Network Latency Metrics for Server Proximity.” In *Proc. Globecom*, San Francisco, CA, Nov. 2000. IEEE.

[16] OMG. “The Common Object Request Broker: Architecture and Specification, revision 2.4.2.” OMG Document Technical Report formal/00-02-33, Object Management Group, Framingham, MA, Feb. 2001.

[17] T. Page, R. Guy, J. Heidemann, R. Ratner, P. Reiher, A. Goel, G. Kuenning, and G. Popek. “Perspectives on Optimistically Replicated, Peer-to-Peer Filing.” *Software – Practice & Experience*, 28(2):155–180, Feb. 1998.

[18] B. Pawlowski, S. Shepler, C. Beame, B. Callaghan, M. Eisler, D. Noveck, D. Robinson, and R. Thurlow. “The NFS Version 4 Protocol.” In *Proc. Second System Administration and Networking Conf. (SANE)*, Maastricht, The Netherlands, May 2000. USENIX.

[19] G. Pierre, M. van Steen, and A. Tanenbaum. “Dynamically Selecting Optimal Distribution Strategies for Web Documents.” *IEEE Trans. Comp.*, 51(6), June 2002. *Scheduled for publication*.

[20] L. Qiu, V. Padmanabhan, and G. Voelker. “On the Placement of Web Server Replicas.” In *Proc. 20th INFOCOM*, Anchorage (AK), Apr. 2001. IEEE.

[21] M. Rabinovich and O. Spastscheck. *Web Caching and Replication*. Addison-Wesley, Reading, MA, 2002.

[22] M. Rabinovich, I. Rabinovich, R. Rajaraman, and A. Aggarwal. “A Dynamic Object Replication and Migration Protocol for an Internet Hosting Service.” In *Proc. 19th Int’l Conf. on Distributed Computing Systems*, pp. 101–113, Austin, TX, June 1999.

IEEE.

[23] S. Rhea, C. Wells, P. Eaton, D. Geels, B. Zhao, H. Weatherspoon, and J. Kubiatowicz. "Maintenance-Free Global Data Storage." *IEEE Internet Comput.*, 5(5):40–49, Sept. 2001.

[24] A. Rowstron and P. Druschel. "Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems." In R. Guerraoui, (ed.), *Proc. Middleware 2001*, volume 2218 of *Lect. Notes Comp. Sc.*, pp. 329–350, Berlin, Nov. 2001. Springer-Verlag.

[25] A. Rowstron and P. Druschel. "Storage Management and Caching in PAST, a Large-Scale, Persistent Peer-to-Peer Storage Utility." In *Proc. 18th Symp. Operating System Principles*, Banff, Canada, Oct. 2001. ACM.

[26] S. Shepler, C. Beame, B. Callaghan, M. Eisler, D. Noveck, D. Robinson, and R. Thurlow. "NFS version 4 Protocol." RFC 3010, Dec. 2000.

[27] M. Spasojevic and M. Satyanarayanan. "An Empirical Study of a Wide-Area Distributed File System." *ACM Trans. Comp. Syst.*, 14(2):200–222, May 1996.

[28] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications." In *Proc. SIGCOMM*, San Diego, CA, Aug. 2001. ACM.

[29] M. van Steen and G. Ballintijn. "Achieving Scalability in Hierarchical Location Services." Technical Report IR-491, Vrije Universiteit, Department of Mathematics and Computer Science, Nov. 2001.

[30] M. van Steen, F. Hauck, G. Ballintijn, and A. Tanenbaum. "Algorithmic Design of the Globe Wide-Area Location Service." *Comp. J.*, 41(5):297–310, 1998.

[31] M. van Steen, P. Homburg, and A. Tanenbaum. "Globe: A Wide-Area Distributed System." *IEEE Concurrency*, 7(1):70–78, Jan. 1999.

[32] H. Yu and A. Vahdat. "Design and Evaluation of a Continuous Consistency Model for Replicated Services." In *Proc. Fourth Symp. on Operating System Design and Implementation*, San Diego, CA, Oct. 2000. USENIX.

[33] B. Y. Zhao, J. Kubiatowicz, and A. D. Joseph. "Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing." Technical Report CSD-01-1141, Computer Science Division, University of California, Berkeley, Apr. 2001.