The Graph Name System: pathnames and petnames in a rootless DNS

Tony Finch
⟨fanf2@cam.ac.uk⟩ ⟨dot@dotat.at⟩
University of Cambridge Computing Service

August 2012

1 Introduction

Names in the Domain Name System are absolute: they specify the complete path from the root to a set of resource records. The DNS has some abbreviation mechanisms: the wire format has name compression, in which trailing labels (the most significant end) of a domain name are represented as a pointer to a previous occurrence of the same labels in the same DNS message; the presentation format in master files represents absolute names with a trailing dot, and if the dot is missing the name is relative to an origin which is typically the name of the zone; and resolvers often have a search mechanism that allows users to leave trailing labels implicit. But these mechanisms are just for convenience, and an abbreviated name must be expanded to an absolute name before it can have any meaning.

What if the DNS had relative names instead?

When an iterative DNS resolver looks up a name, it first sends the query to a root name server, which will return a referral to a top-level domain server; the resolver re-sends the query to the TLD server which will return a referral to a second-level domain server; and so on until the query is sent to a server that can answer it authoritatively. Each time the same query is sent with the same absolute name.

If query names are relative, then when the resolver gets a referral it needs to trim trailing labels off the query name before sending the query to the next server. The trimmed labels correspond to the part of the name in the referring zone, and the remaining query name is relative to the next zone's apex.

But how does the next name server know which zone the query is for? Say the original query was for `www.example.com`. If the resolver trims all the referral labels off, it will send a query for just `www` to an `example.com` name server. This implies a name server can only host one zone, or it needs an IP address per zone. This is not feasible! Another possibility might be to trim off all but one of the labels, so the resolver sends a query for `www.example` to an `example.com` name server. But this means the name server cannot host different zones for `example.com` and `example.org`. Allowing only one zone for each label puts too much pressure on the namespace of single labels, so it isn't feasible either.

DNSSEC provides a solution. As well as a list of name servers (an NS RRset), a secure referral includes a list of digests of the zone's "secure entry point" keys[1] (a DS RRset). We can use one of these digests to identify the zone[2].

[1] The more common term is "key-signing keys", but a zone does not have to have separate KSKs and zone-signing keys. Also, a SEP key usually has its SEP flag set, but this is not required.

[2] There are a few technicalities worth pointing out: When identifying zones by their keys, different zones must have different keys, which is not strictly necessary with standard DNSSEC. A zone can have more than one SEP key, and more than one DS record for each key, so its authoritative servers will have to recognize that any of them can refer to the zone. This implies the choice of algorithm for the DS records must be made by the zone's hostmaster, not by the parent zone's

Resolution then proceeds like this:

- Send a query for `root-DS/www.example.com` to a root name server.
- Receive a referral to the `.com` name servers.
- Send a query for `com-DS/www.example` to a `.com` name server.
- Receive a referral to the `example` name servers.
- Send a query for `example-DS/www` to an `example` name server.
- Receive the answer.

The first consequence of this is that the authoritaive name server is unaware of the path that a resolver took to reach a particular zone. This means there can be multiple paths to a zone: `example.com` and `eg.net` can be the same zone in a stronger sense than in the standard DNS[3]. It is also possible to create loops in the name space: for example, if zone A has a delegation to zone B which in turn has a delegation back to A, then names can have an arbitrarily long sequence of A.B.A.B.A cycles round the loop. (Section 5 explains that loops are normal, not just a gimmick.) Resolvers can traverse the name space starting at any point: they aren't restricted to a unique root.

Thus, instead of being a tree, the namespace becomes an arbitrary graph. Hence we will refer to this modified DNS as the "Graph Name System" or GNS.

Changing the topology of the name space in this way has profound effects on the way the name system can be used. The rest of this paper explores these consequences, and what other changes are needed to make it workable. This is a speculative exercise in the spirit of clean-slate internetwork architecture, so we do not worry about deployability. (For instance, section 6 describes how application protocols need to change the way they use the name system.) But we are not wiping the slate completely clean: the DNS has plenty of limitations (the parochial name syntax), warts (such as the vestigial CLASS selector) and accumulated hacks (a grafted-on extension mechanism; underscrore labels to make up for the lack of per-service selectors) that we will not attempt to fix here.

2 Zooko's Triangle

Zooko Wilcox-O'Hearn argues that names can have two out of three desirable properties[4]. In his introduction to petname systems, Marc Stiegler describes this trade-off in terms of the following properties:

Global The scope of a name's validity crosses trust boundaries. Names can be transferred between users without affecting their meaning.

User-friendly The name is easy for humans to use. It is memorable enough that if you see it written on the side of a bus, you can use it when you get home.

Secure The association between a name and its owner is not vulnerable to interference by some third party.

hostmaster as standard DNSSEC allows.

[3] For instance the RRset signatures will be the same. With standard DNSSEC, zones with the same contents and same keys but different parents will have different signatures, because the absolute names of the records are included in their signatures.

[4] In Zooko's original essay, the properties are User-Friendly, Secure, and Decentralized. These are not independent: a secure name system must be decentralized. Hence we prefer a different trichotomy.

Names in the DNS are **Global** and **User-friendly** but not secure, since the user and publisher of a name have to rely on its delegation chain from the root.

In the GNS a zone has a *formal name* which is its cryptographic identity, as expressed by its DS records, coupled with its location, described by its NS records and the associated address records. Formal names are **Secure** and **Global** but hard to use.

The GNS supports DNS-like *pathnames*. If a pathname starts from a well-known zone (like the DNS root) then it is **Global** and **User-friendly**.

The third possibility is *petnames*, which are **User-friendly** and **Secure** but only valid within a local scope. A GNS resolver has a private zone used to map *petnames* to *formal names*[5].

Thus the GNS supports names that occupy all three sides of Zooko's triangle, although a particular name can have at most two of the three desirable properties.

3 Common DNS problems

There are three common situations which are not well supported by the DNS, and for which there are widely-deployed workarounds. This section describes how they are better supported by the GNS.

3.1 The resolver search path

Many DNS stub resolvers allow users to abbreviate names by omitting trailing labels. The resolver has a "search list" of domains, each of which is appended to the query name in turn until the lookup succeeds. This search process is not very safe, since it depends on which names exist in the domains on the list. RFC 1535 describes a particularly unfortunate example of this lack of safety; the search algorithm was modified to make its behaviour more predictable.

There is no need for a search list in the GNS. Petnames can be used as short aliases for global pathnames, since a zone can have multiple names. The association between a petname and the zone it refers to is secure, so the RFC 1535 problem never occurs.

3.2 Private namespaces

It is common for corporate internal networks to have private namespaces that are not hooked into the public DNS. This can be done using a sub-zone of the corporation's public zone whose nameservers are not accessible outside the corporate network, e.g. `private.example.com`. The sub-zone can be kept secret by omitting its delegation from the public zone, provided the corporate resolvers are configured to know about it. However names of this kind can become uncomfortably long, so sometimes hostmasters configure their resolvers to divert as-yet-unallocated parts of the public namespace for internal use, such as using the `.corp` TLD for internal names, or using `eg.int` (reinterpreting the `.int` TLD to mean "internal" rather than "intergovernmental treaty oranization").

These kinds of hacks work OK in the absence of DNSSEC. However users who have validating resolvers will get a proof of nonexistence from the public name space which prevents resolution of private names. This can be fixed by configuring a trust anchor for the private namespace, though doing so is not easy.

GNS petnames can be used to implement short names for private zones. They require about the same amount of configuration as a DNSSEC trust anchor for a private namespace, i.e. a cryptographic blob and a list of authoritative servers. This can be made simpler using zero-configuration networking mechanisms.

[5] In a true petname system the reverse mapping from keys to names is used so that users are shown petnames instead of cryptographic keys. Section 7 describes how to do this with the GNS.

3.3 Zero-configuration networking

Resolvers expect DNS names to be linked into the global name space. This makes it impossible to support zero-configuration networking, since the zone used to name devices must either have its delegation configured, or resolvers must be configured with a private namespace hack. So unmanaged networks use different name services such as NetBIOS or Multicast DNS.

The GNS can support unmanaged networks as follows:

Each device has a zone containing a description of the services it provides, which it publishes to other devices on the same network. It advertises the existence of this zone using the network's zero-configuration mechanisms, such as a Neighbor Discovery option.

Each device uses these advertisements to maintain zones that contain delegations to other devices' service discovery zones. A device will have one of these zones for each network it is connected to. Each of these zones is given a petname to hook it into the user's namespace. This two-level scheme keps the effects of untrustworthy zone advertisements isolated from trusted petnames.

4 The root zone

The previous sections have glossed over the relationship between local petnames and global pathnames in the GNS. The two need to exist within the same namespace, otherwise names have to carry a local/global flag around. The form of the name itself should indicate which it is, so that a person can communicate the name without computer assistance.

One possibility is to unify Petnames and pathnames by setting up a petname for each global TLD. (This would be done automatically by the user's software - see section 8.) The user's petname zone becomes their personal root zone, the starting point for all their name resolution. Petnames are then short names that do not cross multiple trust boundaries, and pathnames are longer names that do.

The system as a whole has no single root zone that is the basis of a globally unique namespace. Instead of being baked into the name service, the standard namespace becomes purely a matter of policy.

This has some important implications. The root zone is no longer an operational service, so there is no need for root servers as in the DNS[6]. The root zone maintenance agency (the equivalent of ICANN for the DNS) becomes a standards maintenance organization that just curates the list of TLDs. It is less powerful than ICANN, because it depends on software vendors to implement the standard.

However, putting TLDs and petnames in the same zone sets up an unfortunate tussle over which names are TLDs and which names can be petnames. A shared namespace can only work if the set of TLDs is relatively small and very stable, so that the names remain memorable and distinctive. (For instance, what happens when a new TLD is introduced that clashes with a petname?)

It is possible to segregate the namespace, perhaps by reserving single-character TLDs for use by petnames. Entries in the user's root zone would then have the form `eg.Q` for petnames, or `QQ` for country-class TLDs, or longer for generic global TLDs.

It appears to be easier in the GNS to deviate from the standard global namespace than it is with the DNS: for instance a vendor could implement support for unofficial TLDs, or a government could require support for national names. But both of these are possible with the DNS (using the kind of namespace hacks discussed in section 3.2), and might in fact be easier because there are relatively few ISP-wide recursive DNS resolvers compared to the number of personal GNS resolvers. We expect that network effects would strongly encourage people to stick with the standard namespace.

The GNS might make it possible to establish a TLD without going through the standard process. If a service becomes popular enough and manages to persuade enough of its users to configure the same

[6] But there is still a need for a set of name servers with very stable addresses: see section 8.

petname for its zone, then this could become a de facto TLD. But pure registries do not provide any added value beyond ownership of part of the namespace[7] so they are unlikely to become at all popular. So this is probably only feasible for the likes of Facebook and Google that provide useful services, and the resulting TLD is unlikely to work like the ones we are used to.

5 Names in zones

The introduction described what it means for query names to be relative. All other names in the GNS are relative too: In reply messages, names are relative to the apex of the zone identified by the query DS; In zone master files dot-terminated absolute names are no longer permitted. As well as record owner names, this also applies to names in record data, such as `NS` and `MX` target names.

In the DNS it is common for zones to contain names in other zones, for outsourced name servers or mail servers or other similar arrangements. These other zones are usually in a different branch of the hierarchial namespace, and they are identified simply by absolute name. In the GNS the only way to use a name in another zone is via a delegation from the current zone. This means that loops in the namespace graph are common, such as when two zones provide services to each other.

Another consequence is that the zone of a name service or mail service provider will have very large numbers of delegations from customer zones. It will be practically impossible to keep them all up-to-date, so the GNS needs a mechanism to automatically update delegations. (Arguably the DNS needs one too, because lame delegations are a common problem.)

We make a distinction between "official" and "unofficial" delegations. In an official delegation, the child zone is relying on its parent for the purpose of providing a desirable name (see section 7) or for disaster recovery (see section 8); the owner of the child probably pays the parent, and there is likely to be some out-of-band mechanism to approve changes. In an unofficial delegation, the parent is relying on its child to provide a service, usually related to a higher-level application; the child may receive money from the parent, but might not know about the parent at all. In particular, the entries in a resolver's root zone are usually unofficial delegations. Unofficial delegations must be updated automatically.

In the DNS it is convenient to have out-of-zone name servers, since then you don't need to include "glue" records (name server addresses) in the delegation. The disadvantage is slower resolution times and an increased number of trusted third parties. In the GNS out-of-zone name servers are less convenient, since they require additinal delegations in parent zones.

References to outsourced services can be more secure in the GNS than the DNS. A delegation uses a zone's *formal name*, so if the customer's unofficial delegation refers directly to its provider's zone, it can use outsourced service names without relying on any third parties. In the following abbreviated example, the less secure version refers to the mail service provider by its usual name (similar to what you would do in the DNS), which requires an unofficial delegation to a TLD zone; the more secure version refers directly to the mail service provider's zone so does not need to trust the TLD.

```
; less secure

@       SOA  ( ... )
        MX   mx.example.net.x

net.x   DS   ( ... )
        NS   a.gtld-servers.net.x
```

```
; more secure

@           SOA  ( ... )
            MX   mx.example.x

example.x   DS   ( ... )
            NS   ns1.example.x
```

[7] The split registry/registrar economic model requires registries to be rentiers since only registrars can provide user services.

6 Application names

Application protocols that use DNS names assume that an particular instance of the application (such as a mail domain or a web site) can be identified by a single name, perhaps with a few known aliases. This is not true in the GNS, which encourages users to create unofficial aliases (*petnames*) that the application does not know about.

There are a couple of ways to deal with this difficulty. One alternative is to work around it using the canonical name mechanism described in the next section. This section describes a more secure clean-slate approach.

As described in the introduction, a GNS client maps aliases of zones to a cryptogaphic identifier, which it passes to the server as part of the request. This interaction is self-authenticating.

7 Canonical names

In the following abbreviated example, the `.com` zone is identified by a key with tag 5678[8]; it has a name server `a.ns.com` and delegation to example.com with glue for `ns1.example.com`. "`@`" refers to the zone apex.

The `example.com` zone is identified by a key with tag 1234; its name server information matches the delegation in the `.com` zone. It has a `SELF` record which refers to its delegation in the `.com` zone. In order that it can refer to a name inside the `.com` zone, the `example.com` zone has an unofficial delegation to it named `com.x` (or in full, `com.x.example.com`).

```
; in the .com zone

@             SOA     ( ... )
              DNSKEY  ( ... ) ; tag 5678

              NS      a.ns
a.ns          A       ( ... )
              AAAA    ( ... )

example       DS      1234 ( ... )
              NS      ns1.example
ns1.example   A       ( ... )
              AAAA    ( ... )
```

```
; in the example.com zone

@             SOA     ( ... )
              DNSKEY  ( ... ) ; tag 1234
              SELF    1  example.com.x
              NS      ns1
ns1           A       ( ... )
              AAAA    ( ... )

com.x         DS      5678 ( ... )
              NS      a.ns.com.x
a.ns.com.x    A       ( ... )
              AAAA    ( ... )
```

8 Bootstrap and recovery

9 Summary

The previous sections have described the GNS piecemeal, in the course of exploring how it might be used. Here we collect together the differences between the DNS and the GNS.

The GNS requires DNSSEC.

[8] DNSSEC key tags are 16 bit hashes which make it easier to find the DNSKEY record that was used to generate an RRSIG or DS record.

All names in a GNS zone are relative to the zone apex. All references to names in other zones must be via a delegation in the current zone.

GNS queries and replies include a DS RDATA field (the record without its owner name or signature) before the question section. All names in the rest of the message are relative to the apex of the zone identified by the DS record.

When a resolver receives a referral response (containing DS records, NS records, and the associated address records) it chooses a DS record with which to update the query. The DS owner name is stripped off the right-hand end of the query name, and the DS RDATA replaces the query DS.

Each resolver maintains its own root zone, which contains the standard TLDs and the user's petnames. An allocation policy keeps the two kinds of name separate, for example, reserving single-letter TLDs for petnames. This root zone defines the user's known namespaces and associated trust anchors.

Most zones (including resolver roots) contain unofficial references to other zones, which are not maintained by the owner of the other zone. The name server must have a mechanism to maintain them automatically,

A zone may contain `SELF` records at its apex. These have the same format as MX records: a 16 bit priority field and a target name. The name must traverse a delegation from another zone to the zone itself.

10 Conclusion