How to Announce Parent “Services” to Children?

Johan Stenstam

November 10, 2023
Separating the Issues

In the discussion there seems to be some mixing of the **delivery mechanism** for “a service” that a parent may want to announce:

- a generalized NOTIFY (to the benefit of a scanner)
- a DNS UPDATE Receiver (when there is no scanner)
- a RESTful API endpoint (when there is a separate provisioning infrastructure to use)
- etc.

and **how the service is located**:

- New record(s) in the parent zone, in an SVCB-like RR type.
- SOA.MNAME (that’s where UPDATE usually go)
- Parent NS RRset (because that’s where traditional NOTIFIES go)
- etc.
It may be useful to take a step back and focus on one thing at a time.

The problem is that this is not how the two documents are written.

- They are written as “generalized NOTIFY and how to know where to send it” and “DNS UPDATE and how to know where to send it”

Let’s try to mentally restructure this. I therefore propose the hypothetical service “change the TTL on my delegation records”. This service is only available via the registrar’s REST API endpoint on:

```plaintext
https://api.registrar:9053/api/v2/
```

using version 2 of a hypothetical provisioning protocol.

- How should the parent announce availability of this service and where to reach it?
Here is one suggestion: The NOTIFY draft uses a parameter called scheme to indicate different methods to “communicate with the parent”:

```
child._dnsync.parent. IN DSYNC ANY 2 5310 ddns-updates.parent.
```

Strawman proposal for announcing “API access”:

```
child._dnsync.parent. IN DSYNC ANY 3 9053 api-spec.registrar.
api-spec.registrar. IN SRV 10 10 9053 api.registrar.
api-spec.registrar. IN TXT registrar-dns-provisioning-api-v2
```

- “2=UPDATE” Where to send DNS UPDATE
- “3=API” Where to find API details
- API server (the “target”) API specification
Automating DNS Parent—Child Synchronization

Johan Stenstam

November 10, 2023
Automating DNS Parent—Child Synchronization via DNS UPDATE

Johan Stenstam

November 10, 2023
Problem Statement

Since the inception of time

- i.e. when DNS was invented and life as we know it begun there has been a problem with the delegation information (the NS RRset and sometimes the glue records) getting out of sync between the (authoritative) data in the child zone and the delegation in the parent zone.

- Slowly, over time, updates happen that are not synched with the parent. For all sorts of reasons.

- We’re beginning to make some inroads on automatic synchronization for a subset of parent/child cases via so-called CDS and CSYNC scanners run by the parent.

- But most of the DNS name space is outside of that subset
What Are The Use Cases?

While the risk of “parent and child getting out of sync” exists for all zone cuts, the risk is clearly not the same everywhere.

- Some registries do periodic, proactive scans of all delegations, trying to catch errors before they cause problems.
- “Corporate environments” in many cases have all zone management under control of their IPAM-systems, that can keep things in sync.

So perhaps there is no problem to be solved?
Use Cases, cont’d

But even so, there are lots of cases when zone cuts cross organisational borders and are therefore more difficult to maintain.

- Academia and the educational sector in general.
- The health care sector: lots of hospitals with lots of departments.
- Government agencies and departments.
- A gazillion zones that are important to the owner, but DNS is not a focus, DNSSEC isn’t used and the parent is not proactive.

In short, many, many places where DNS is not a core focus, nor a core expertise.

It would be great if we could find a way to keep such delegations in sync automatically.

... and even better if it could be done with trivial (or possibly even zero) configuration in the child end.
But even so, there are lots of cases when zone cuts cross organisational borders and are therefore more difficult to maintain.

- Academia and the educational sector in general.
- The health care sector: lots of hospitals with lots of departments.
- Government agencies and departments.
- A gazillion zones that are important to the owner, but DNS is not a focus, DNSSEC isn’t used and the parent is not proactive.

In short, many, many places where DNS is not a core focus, nor a core expertise.

- It would be great if we could find a way to keep such delegations in sync automatically.
- ... and even better if it could be done with trivial (or possibly even zero) configuration in the child end.
Let’s Talk About Scanners

CDS and CSYNC scanners are “pull based”. This is not as efficient as a “push based” mechanism. That is being addressed via the ongoing work on generalized notifications.

However, there are other issues with scanners apart from efficiency.

- **Scanners require** the children to have **DNSSEC** deployed.
- **Scanners are complex.** It is yet another service to run and maintain. They require complicated logic.
  - It seems unlikely that scanners will ever be a popular choice outside of the registry space.
  - I.e. they represent a partial solution that only work for some, not for all zones. That’s always bad.

But, given their drawbacks, scanners do work, and that’s why a growing number of TLDs (and some registrars) are deploying scanners.
Synchronisation Without A Scanner?

Could it be possible to find a design where we get the automated synchronisation between child and parent without the parent having to operate a scanner?

Yes.

Such a design already exists, but has not seen much use yet. It’s based on UPDATE (i.e. DNS Dynamic Updates).

- However, it is not based on UPDATE using TSIG keys (that’s a shared secret system and therefore doesn’t scale well to large numbers of children).
- Instead it is based on using so-called SIG(0) signatures for the UPDATE for security.
- An additional feature is that the SIG(0) validation mechanism works independently of DNSSEC.
Dynamic Update Policies For Many Delegations

Assume delegations that use in-bailiwick nameservers, a la

```
foo.parent. IN NS ns1.foo.parent.
foo.parent. IN NS ns2.foo.parent.
ns1.foo.parent. IN A 1.2.3.4
ns1.foo.parent. IN AAAA 2001:a:bad:cafe::53
ns2.foo.parent. IN A 2.3.4.5
```

In one implementation (BIND9) it is then possible to use an update policy a la:

```
update-policy {
    grant * selfsub * NS A AAAA KEY ;
};
```
Dynamic Update Policies For Many Delegations

Assume delegations that use in-bailiwick nameservers, a la:

```
foo.parent. IN NS ns1.foo.parent.
foo.parent. IN NS ns2.foo.parent.
ns1.foo.parent. IN A 1.2.3.4
ns1.foo.parent. IN AAAA 2001:a:bad:cafe::53
ns2.foo.parent. IN A 2.3.4.5
```

In one implementation (BIND9) it is then possible to use an update policy a la:

```
update-policy {
    grant * selfsub * NS A AAAA KEY;
}
```

For a match, both wildcards must expand to the same string.

Only these RR types.

Exact match and subdomains of match.

I.e. it is impossible to affect any delegation except "your own".
Dynamic Update Policies For Many Delegations

Assume delegations that use in-bailiwick nameservers, a la

```
foo.parent. IN NS ns1.foo.parent.
foo.parent. IN NS ns2.foo.parent.
ns1.foo.parent. IN A 1.2.3.4
ns1.foo.parent. IN AAAA 2001:a:bad:cafe::53
ns2.foo.parent. IN A 2.3.4.5
```

In one implementation (BIND9) it is then possible to use an update policy a la:

```
update-policy {
  grant * selfsub * NS A AAAA KEY ;
}
```

This policy would allow a key with the name `foo.parent.` to update records like `foo.parent. NS ...` and `ns1.foo.parent. A ...` but not any records that do not have an owner name ending in `foo.parent.`.

- i.e. it is impossible to affect any delegation except “your own”.

For a match, both wildcards must expand to the same string

Only these RR types

Exact match and subdomains of match
DNS UPDATE? Are you mad?

According to my wife and our dog this is most likely.
DNS UPDATE? Are you mad?

According to my wife and our dog this is most likely.

- But the dog always takes her side, so that doesn’t really count.
DNS UPDATE? Are you mad?

According to my wife and our dog this is most likely.

- But the dog always takes her side, so that doesn’t really count

But bear with me for a bit.

There is lots of prior art here. For instance the (expired) draft-andrews-dnsop-update-parent-zones-04 describes exactly the same idea.

- But it never took off. The question is why?

The question of why UPDATE isn’t already used for delegation synchronization is really the key question.

- After all, the functionality has been there for years.

I argue that the reason is a combination of two factors:
Problems Using DNS UPDATE For Synchronisation

- **Problem #1**: It is difficult to figure out where to send the DNS UPDATE. UPDATE is mostly used inside an organisation, but in the parent/child case there is often a need to cross an organisational boundary.
  - The parent primaries are often hidden, but even if it is known, it may be difficult to reach because it is locked down behind firewalls, etc. This would require “update forwarding”, which is icky.

- **Problem #2**: The assumption has been that the UPDATE is sent to a nameserver, and, if accepted, is immediately applied to the zone.
  - This is not acceptable to many parent zones. They have requirements on applying policies and safety checks (not to mention audit trails) on any data before it is added to the zone.

But... both of these issues are addressed in the Internet-Draft about generalised notifications. Let’s combine the two!
Combining Generalised Notifications and DNS UPDATE

The notification draft uses a parameter called scheme to indicate possible different methods to “communicate with the parent”:

```
parent. IN SOA ...
...
parent. IN NOTIFY CDS 1 5301 notifications.parent.
parent. IN NOTIFY CSYNC 1 5302 notifications.parent.
```

Let’s add a new scheme for the UPDATE-based synchronization:

```
parent. IN NOTIFY ANY 2 5310 ddns-updates.parent.
```
Combining Generalised Notifications and DNS UPDATE

The notification draft uses a parameter called `scheme` to indicate possible different methods to “communicate with the parent”:

```
parent. IN SOA ...
...  
parent. IN NOTIFY CDS 1 5301 notifications.parent.
parent. IN NOTIFY CSYNC 1 5302 notifications.parent.
```

Let’s add a new scheme for the UPDATE-based synchronization:

```
parent. IN NOTIFY ANY 2 5310 ddns-updates.parent.
```

Where to send the NOTIFY

Where to send the UPDATE

Port for DNS UPDATE
Combining Generalised Notifications and DNS UPDATE

The notification draft uses a parameter called **scheme** to indicate possible different methods to “communicate with the parent”:

- \texttt{parent. IN SOA ...} ... \texttt{parent. IN NOTIFY CDS 1 5301 notifications.parent.} \texttt{parent. IN NOTIFY CSYNC 1 5302 notifications.parent.}

Let’s add a new scheme for the UPDATE-based synchronization:

- \texttt{parent. IN NOTIFY ANY 2 5310 ddns-updates.parent.}

That’s works, but… there is a naming confusion here.

- The mnemonic **NOTIFY** doesn’t work well for DDNS updates.
- But what should be used instead? Not my decision, but let me just throw out a suggestion, based on the topic being “delegation synchronisation”.

Johan Stenstam
Automating DNS Parent—Child Synchronization
November 10, 2023 15 / 26
Combining Generalised Notifications and DNS UPDATE

The notification draft uses a parameter called **scheme** to indicate possible different methods to “communicate with the parent”:

```
parent. IN SOA ... ... 
parent. IN DSYNC CDS 1 5301 notifications.parent.
parent. IN DSYNC CSYNC 1 5302 notifications.parent.
```

Let’s add a new scheme for the UPDATE-based synchronization:

```
parent. IN DSYNC ANY 2 5310 ddns-updates.parent.
```

That’s works, but… there is a naming confusion here.

- **DSYNC** would work well for both NOTIFY and UPDATE.
- But what should be used instead? Not my decision, but let me just throw out a suggestion, based on the topic being “delegation synchronisation”.

Johan Stenstam  
Automating DNS Parent—Child Synchronization  
November 10, 2023 15 / 26
Where To Send The NOTIFY or UPDATE

The presentation on Tuesday only dealt with this issue and my summary is:

- We must use the same algorithm for locating the target (and mechanism) for both NOTIFY and UPDATE (and possible other mechanisms)
- There are two issues: which RR type and which qname to query for
- There seem to be agreement that the best thing is a new RR type with the same wire format as SVCB.
- There have not been any objections (nor support) to our suggestion to first query for `child._dsync.parent. DSYNC` and if no data there then, as a fallback, query for `parent. DSYNC`.
  - This still seems like the best alternative to us
Let’s Look At the Different Cases

<table>
<thead>
<tr>
<th>Types of parents</th>
<th>Signed children</th>
<th>Unsigned children</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Registries”</td>
<td><strong>OK</strong></td>
<td>?</td>
</tr>
<tr>
<td>Other parents</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

An important detail is that the SIG(0) keys **do not have to be present in the parent zone**. That’s an artifact of the BIND9 implementation.

- The SIG(0) public keys only have to be available to the UPDATE Receiver.
## Let’s Look At the Different Cases

<table>
<thead>
<tr>
<th>Types of parents</th>
<th>Signed children</th>
<th>Unsigned children</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Registries”</td>
<td>OK</td>
<td>?</td>
</tr>
<tr>
<td>Other parents</td>
<td>OK</td>
<td>?</td>
</tr>
</tbody>
</table>

An important detail is that the SIG(0) keys do not have to be present in the parent zone. That’s an artifact of the BIND9 implementation.

- The SIG(0) public keys only have to be available to the UPDATE Receiver.

DNS UPDATE (but, modulo complexity, scanners would also work)
Let's Look At the Different Cases

<table>
<thead>
<tr>
<th>Types of parents</th>
<th>Signed children</th>
<th>Unsigned children</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Registries”</td>
<td>OK</td>
<td>?</td>
</tr>
<tr>
<td>Other parents</td>
<td>OK</td>
<td>OK</td>
</tr>
</tbody>
</table>

An important detail is that the SIG(0) keys **do not have to be present in the parent zone**. That’s an artifact of the BIND9 implementation.

- The SIG(0) public keys only have to be available to the UPDATE Receiver.

---

DNS UPDATE FTW (I see no other solution)
Let’s Look At the Different Cases

### Types of parents

<table>
<thead>
<tr>
<th>Signed children</th>
<th>Unsigned children</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Registries”</td>
<td>“SOL”</td>
</tr>
<tr>
<td>Other parents</td>
<td>OK</td>
</tr>
</tbody>
</table>

An important detail is that the SIG(0) keys do not have to be present in the parent zone. That’s an artifact of the BIND9 implementation.

- The SIG(0) public keys only have to be available to the UPDATE Receiver.

DNS UPDATE (but, modulo complexity, scanners would also work)

This is an interesting problem space

DNS UPDATE FTW (I see no other solution)
Could All CDS/CSYNC Scanning Be Replaced By DNS UPDATE?

This sounds like a technical question but it really isn’t. It’s a market question.

- From a technical POV, UPDATE would work fine in all cases, including parents that are registries.
  - The UPDATE could be sent to the registrar to be transformed into an EPP transaction.
- But that would require changes in the DNS market. Changes cost money. I am skeptical that DNS UPDATE are sufficiently “better” than scanners + NOTIFY to drive such changes.
- The focus here is primarily on all the zone cuts in the rest of the DNS hierarchy.
Let’s Talk About Efficiency. And Simplicity

What about UPDATE-based synchronization (regardless of the type of parent)?

- UPDATE is **efficient**, **converge fast** and **does not require DNSSEC**.

But what about the complexity? Isn’t an UPDATE Receiver about as complex as a scanner?

- No it isn’t.
- The Receiver is essentially only the policy verification and subsequent action of a scanner without the rest of the scanner (and no rate-limiting issues, retries, etc)
- After approval of the UPDATE, the Receiver could emit a series of “add RR, delete RR, delete RR, add RR, add RR, delete RR, . . .” instructions towards whatever backend provisioning is used for the parent zone.
Let’s Summarise

- The “NOTIFY scheme” and the “UPDATE scheme” are alternate methods for parent synchronisation.
- The same mechanism is used for signalling support for generalised NOTIFY (and where to send them) and support for UPDATE (and where to send them).
  - This addresses the first major issue with using UPDATE for child-to-parent synchronization.
- By sending the UPDATE to a “service” rather than to the parent primary nameserver, the parent may implement whatever policies and checks that it wants before applying the UPDATE to the datastore the parent zone is generated from.
  - This addresses the second major issue with using UPDATE updates across organizational boundaries.
  - This also removes the requirement that the parent primary has to allow dynamic updates.
Summary

<table>
<thead>
<tr>
<th>Types of parents</th>
<th>Signed children</th>
<th>Unsigned children</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Registries”</td>
<td>NOTIFY+Scanner</td>
<td>“SOL”</td>
</tr>
<tr>
<td>Other parents</td>
<td>UPDATE</td>
<td>UPDATE</td>
</tr>
</tbody>
</table>

- Using the mechanism for locating the message target from the generalised notifications...
- and separating parent verification from parent zone update...
- combined with SIG(0)-authenticated DDNS updates...
- enables an efficient and scalable solution to the issue of automatic synchronisation of delegation information.

The method has the added advantages of being less complex than CDS/CSYNC scanners and also work independently of DNSSEC.
How Should The SIG(0) Public Keys Be Distributed?

This is obviously an important issue.

- The child `foo.parent.` must have access to both the private and the public key with the name `foo.parent.`
- The UPDATE Receiver in the parent end must have access to the public key `foo.parent.`

While this problem is central, it is important to be aware that it may be only a **bootstrapping issue**. Once the key is known to the parent, the child could initiate a rollover of the SIG(0) key via DNS UPDATE.

- There is no difference between an UPDATE to change the NS RRset or an UPDATE to change the KEY RRset.
- On the other hand, in most cases, not rolling the key will be sufficient.
In the case where the child zone delegation is signed this is a rather simple problem with at least two workable solutions:

- Look up and validate the `child.parent. KEY...` every time and use that.
- Look up and validate the child KEY the first time and cache it in a local keystore.

The advantage of caching the key is that it makes subsequent UPDATE validations independent of DNSSEC validation. The downside is that then there needs to be some mechanism to invalidate the cached key (e.g. sending an UPDATE which deletes that key, forcing a re-fetch and DNSSEC validation of the new key).
Distribution of SIG(0) Public Keys: Unsigned Case

There are various different alternatives:

- Provision the public key as part of the initial delegation. For non-registries this is usually a somewhat manual process.
- Add a method for uploading the public key via an authenticated web portal.
- Communicate the public key to the parent via whatever mechanism is used to communicate changes to the NS RRset or glue.

There is no magic bullet in the unsigned case, but every unsigned child zone has somehow managed to communicate enough information to the parent to set up the delegation.

- This mechanism has to be used one final time to communicate the public SIG(0) key to the parent DNS UPDATE Receiver.
WG Adoption of this Document?

The only thing that is **new** with this document is that we **combine two things** that are either already done, or already in the WG:

- the well-known and widely used DNS UPDATE mechanism for communicating changes from a sender to a receiver ([RFC2136](https://tools.ietf.org/html/rfc2136), [RFC3007](https://tools.ietf.org/html/rfc3007))

with

- the method to locate the target of the DNS UPDATE shared with the WG document on generalized NOTIFY ([draft-ietf-dnsop-generalized-notify](https://tools.ietf.org/dokid/837))

Nothing else.

I would like to see this document adopted by the WG.
<table>
<thead>
<tr>
<th>Johan Stenstam</th>
<th><a href="mailto:johan.stenstam@internetstiftelsen.se">johan.stenstam@internetstiftelsen.se</a></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Working code</strong></td>
<td><a href="https://github.com/johanix/gen-notify-test">https://github.com/johanix/gen-notify-test</a></td>
</tr>
</tbody>
</table>