Reverse DNS What it is and how to configure it in BIND

Greg Choules

2023-10-19

https://www.isc.org

Firstly, what is forward DNS?

Forward DNS takes a name and gives you an address. For example:

```
dig www.example.com
...
www.example.com.   IN   A   1.2.3.4
```

A recursive DNS server navigates the DNS namespace from right to left (domains are the wrong way round!) parsing first "." (or the root, which comes after "com" and is implied), then "com", then "example", then "www" and finally obtaining the answer 1.2.3.4

Firstly, what is forward DNS? (2)

Forward DNS takes a name and gives you an address.
For example:

```
dig www.example.com
...
www.example.com.   IN   A   1.2.3.4
```

A recursive DNS server navigates the DNS namespace from right to left (domains are the wrong way round!) parsing first "." (or the root), then "com", then "example", then "www" and finally obtaining the answer 1.2.3.4
Name to address mappings are stored in A or AAAA records.

So, what is reverse DNS?

Reverse DNS takes an address and gives you a name.
For example:

```
dig -x 1.2.3.4
...
4.3.2.1.in-addr.arpa.   IN   PTR    www.example.com.
```

The address (IPv4 in this case) has been flipped around and ".in-addr.arpa" has been added to it.
As before, a recursive DNS server navigates the DNS namespace from right to left, parsing first ".", then "arpa", then "in-addr", then "1", "2", "3" and "4" and finally obtaining the answer "www.example.com".

Address to name mappings are stored in PTR records.

But what is actually going on here?

Several concepts are being combined in these examples. They are:

- Domains: left to right or right to left?
- The dot character; "."
- "in-addr.arpa"

So let's explore each of them in turn.

Domains: left to right or right to left?

The DNS namespace is a hierarchical tree structure, like a filing system or a phone number, with all names emanating from the root.

In Linux, for example, the top level is called the root and directories below that are separated with the "/" character. The full path to a file might look like this:

```
/users/me/documents/presentation.pdf
```

where the highest, or most generic, level of the hierarchy is to the left and successive "/" take you deeper into the filing system, getting more specific, until you reach the file of interest.

Domains (2)

Telephone numbers don't use a separator character but the hierarchy is implied, again with the most generic or highest level of the hierarchy on the left.

The position and number of digits determines their meaning as you traverse from left to right. For example:

```
+442071234567
```

This shows a UK number (44) in central London (207) with a local area code of 123 and a subscriber number of 4567.

Domains (3)

Similarly, IP addresses are written with the highest order part on the left.

```
192.168.1.2
```

In DNS, though, domains are written with the most generic part (the root) on the right and get more specific as you read to the left. For example:

```
www.example.com(.)
```

The "." at the end of "com" represents the cut before the root and is in parentheses because it is implied. The root itself is the null label "", so does not appear.

The dot character

ASCII 46 (dec; 2E hex), full stop, period, decimal point was chosen as the separator for the human-readable forms of both IPv4 addresses and DNS domains. This matters when it comes to representing IPv4 addresses in DNS names as an IPv4 address can be viewed already as a set of DNS labels.

Interestingly, the "." does not exist in the wire formats of either protocol. IPv4 addresses are transmitted as big-endian 32-bit integers and DNS names are transmitted as a set of labels/lengths. The dots are only there when we write them down!

in-addr.arpa

"arpa" is a Top-Level Domain (TLD) and "in-addr" is a Second-Level Domain (SLD) under it. Both are administered by ICANN and the IAB for infrastructure.

From RFC1035: to create a reverse DNS entry for an IPv4 address, take part of the address, reverse it, append ".in-addr.arpa" to it and use that name for a new zone.

For example, if the addresses to map are 192.168.1.1-254 it would make sense to create the zone:

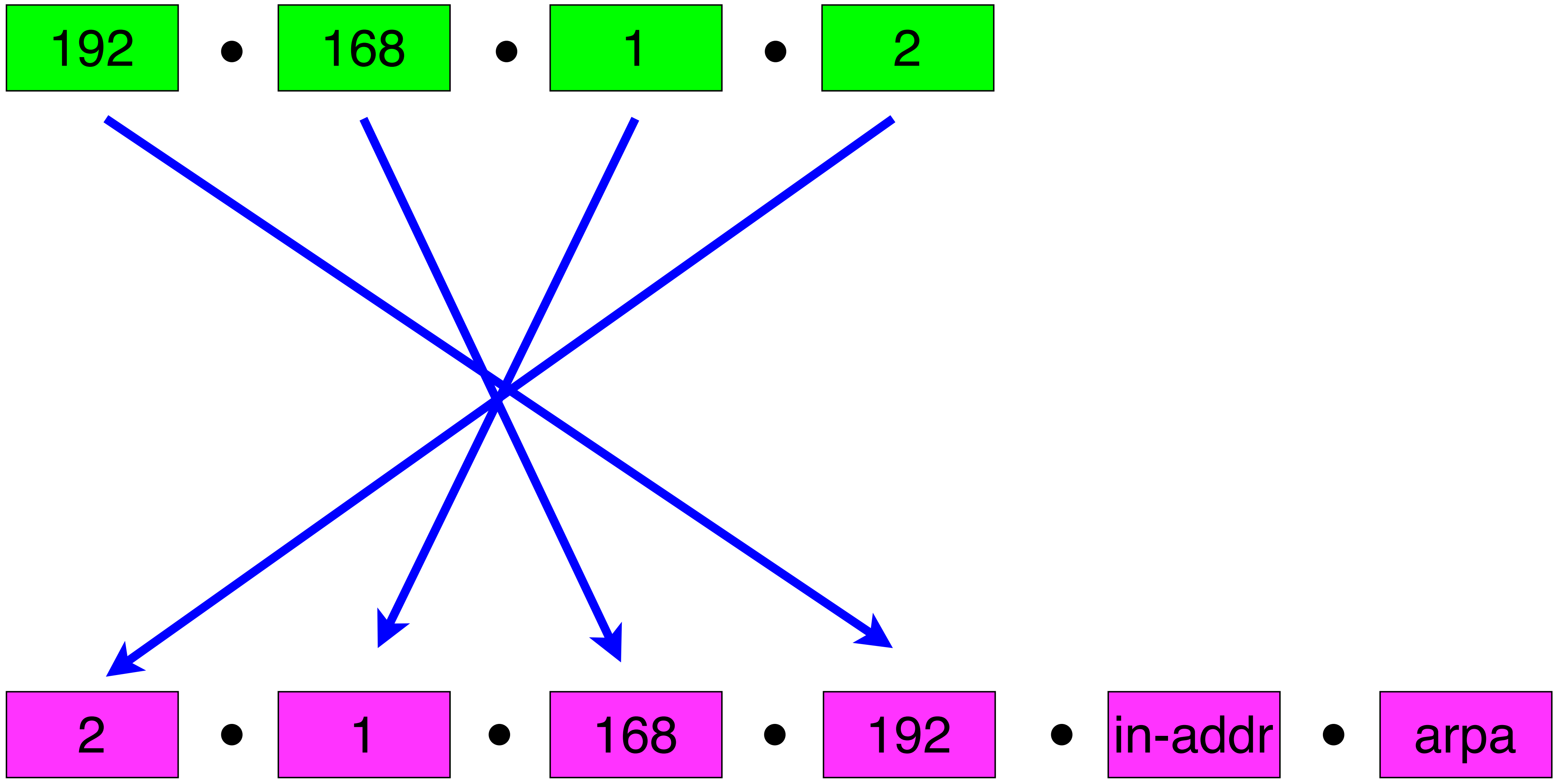
```
1.168.192.in-addr.arpa
```

in-addr.arpa (2)

In this zone, individual addresses appear as owners of PTR records:

```
...
1   PTR   192.168.1.1.mydomain.com.
2   PTR   192.168.1.2.mydomain.com.
3   PTR   unused.
4   PTR   printer1.myinternaldomain.net.
...
```

Mapping



Limitations

One limitation of this method arises because IPv4 addresses are in groups of 8 bits (known as octets) separated with ".". When turned into a DNS name each "." becomes a cut point, creating additional levels in the namespace.

This naturally limits reverse DNS zones to octet boundaries because that's where the dots are. These are valid reverse zones:

```
10.in-addr.arpa
16.172.in-arpa
67.168.192.in-addr.arpa
```

Limitations (2)

In today's Internet, subnetting is done at any number of bits, not just multiples of 8. What happens if you want to create a single reverse zone for this network?

```
192.168.32.0/20
```

The split between network and host bits occurs midway through octet 3 - "32". There is no "." character here, so it cannot be readily mapped to DNS.

RFC2317 suggests a way this can be done. But it is more complex than having zones on octet boundaries. For this example, 16 /24 zones would work well.

How to create reverses in BIND

To make a working zone you need two things:

- Configuration of the zone itself.
- Data for that zone.

A reverse zone is the same as any other zone, but with slightly different contents.

Configuration of a reverse zone might look like this:

```
zone "16.172.in-addr.arpa" {
  type primary;
  file "db.172.16";
};
```

How to create reverses in BIND (2)

Next we need the zone data itself. The zone file is constructed just like any other, but containing PTR record types.

Here is what the zone file for the 16.172... zone might look like:

```
$TTL 3600
@   SOA   mname   rname   1   1800   900   604800   86400
@   NS   ns1.mydomain.net.
1.1   3600   IN   PTR   host-172.16.1.1.mydomain.net.
2.1   3600   IN   PTR   host-172.16.1.2.mydomain.net.
...
255.3   PTR   host-172.16.3.255.mydomain.net.
...
```

How to create reverses in BIND (3)

A few notes about these records might be in order:

- Since the zone has been defined as a /16 (2 octets) the owner field for each record must comprise the other two octets as shown, with separating "."
- In the zone data these last two octets must be reversed. So address 172.16.1.2 becomes 2.1 in this zone. This is logical because the zone name will be appended to the owner, giving 2.1.16.172.in-addr.arpa.
- Unless each /24 zone is also declared in the config there will be an ENT for each different 3rd octet. This may matter if the zones are DNSSEC signed.

Testing reverses

Testing that reverse zones and records work as expected can be done easily with the usual tools. e.g. here is a query for 172.16.1.2 using dig:

```
dig 2.1.16.172.in-addr.arpa PTR
```

This is slightly cumbersome as you have to remember to reverse the IP address first. However, dig provides a handy option to avoid this. It sets the query type to PTR as well:

```
dig -x 172.16.1.2
```

Why do we need reverse DNS?

Very few applications need reverse DNS these days but it can be useful in some circumstances. Two examples of applications that use it are:

- traceroute
- tcpdump

traceroute

Traceroute is a handy tool for discovering and printing each hop on a network path. For example:

```
$ traceroute www.isc.org
traceroute to isc.map.fastlydns.net (151.101.42.217), 64 hops max, 40 byte packets
 1  r2.prod.oak1.isc.org (149.20.2.3)  0.205 ms  0.184 ms  0.188 ms
 2  gigabitethernet1-1-44.switch55.fmt2.he.net (66.160.158.245)  137.708 ms  112.535 ms *
 3  port-channel6.core4.sjc2.he.net (184.105.213.157)  10.969 ms *  4.506 ms
```

For each hop it performs a reverse DNS query on the IP address of that hop (which just returned ICMP hop count expired) so that it can display to the user the name of that hop.

Beware, though, that DNS does not always have an answer, which will give the familiar "* * *" output.

tcpdump

Tcpdump is a command line tool for capturing, decoding and displaying network traffic.

By default it will perform reverse DNS lookups on IP addresses in packets, to display endpoint names. For example:

```
15:38:35.503717 IP6 2a00:23c6:f103:3101:69cd:41c1:facb:5f68.53243 > gitlab.isc.org.https:
Flags [P.], seq 312:433, ack 635, win 2048, options [nop,nop,TS val 25480311 ecr 3845002103],
length 121
15:38:35.593605 IP6 gitlab.isc.org.https > 2a00:23c6:f103:3101:69cd:41c1:facb:5f68.53243:
Flags [.], ack 433, win 473, options [nop,nop,TS val 3845004616 ecr 25480311], length 0
```

This can be useful, but it adds overhead to the capture process. When capturing at high packet rates this can affect performance.

IPv6 and reverse DNS

Reverse DNS lookups also work with IPv6 addresses. The domain and data format are a little different from the IPv4 usage, but the principle is the same.

The domain suffix in this case is "ip6.arpa".

Since IPv6 addresses are in hexadecimal with hextets separated by the ":" character there is no natural fit with DNS domains. So the decision was to make each hex character a separate domain. For example:

```
dig gitlab.isc.org AAAA
gitlab.isc.org.         356     IN     AAAA     2600:1f18:634c:d100:acdc:70e4:ede4:ef40

dig -x 2600:1f18:634c:d100:acdc:70e4:ede4:ef40
0.4.f.e.4.e.d.e.4.e.0.7.c.d.c.a.0.0.1.d.c.4.3.6.8.1.f.1.0.0.6.2.ip6.arpa. 300 IN PTR
gitlab.isc.org
```

Some words of warning

When creating reverse zones you may have the choice of one or a few /24s, /16s or /8s; leaving aside RFC2317. But be careful about what you create versus what you need.

If you use 10/8 internally (for example) it is tempting to create a single zone 10.in-addr.arpa, even if you only use a fraction of that address space.

Or worse, if you are assigned a public /20 (say) it is again tempting to create a single reverse zone that is the /16 containing your allocation.

Some words of warning (2)

For the internal case, creating 10/8 works as long as you are the only DNS in town; e.g. Microsoft (in an AD environment). But if you run many different DNS servers for different purposes, decide on a hierarchy, make the 'parent' DNS 10/8 and remember to delegate.

For the public case, since a registry has assigned you just a /20 you must only be authoritative for that /20, or components of it. In practice it would be safer (and not much work) to create sixteen /24 reverse zones and delegate them, or a subset, from the registry to your authoritative servers.

Thank you
:)

Queries?