# The case of IPv6 LLU endpoint address support in DNS

Erwin Hoffmann
Frankfurt University of Applied Sciences
Faculty of Computer Science and Engineering

January 12, 2020

**Abstract**

The `fehQlibs` and **djbdnscurve6** provide a DNS library which support IPv6 LLU endpoint addresses. The inclusion of IPv6 LLU endpoint addresses for DNS services is discussed and shown to be beneficial for particular use cases. Applications linked with this library allow user-specific DNS settings. Apart from binding to IPv6 LLU addresses and dual-stack usage, additionally 'reverse IPv6 anycasting' is an achieved goal welcomed for IoT devices.

## 1 IPv6 in DNS

The support of IPv6 addresses within DNS Resource Records (RR) has been settled with RFC 3596 [30] taking only the Quad-A format `AAAA` into consideration. The inverse nibble format for IPv6 addresses is also well defined.

However, there is a significant gap applying IPv6 addresses within the DNS, though RFC 4472 [5] clearly expels IPv6 *Link-Local Unicast* (LLU) addresses as AAAA RRs in a zone file:

> 2.1. Limited-Scope Addresses
>
> The IPv6 addressing architecture [RFC4291] includes two kinds of local-use addresses: link-local (fe80::/10) and site-local (fec0::/10). The site-local addresses have been deprecated [RFC3879] but are discussed with unique local addresses in Appendix A.
>
> Link-local addresses should never be published in DNS (whether in forward or reverse tree), because they have only local (to the connected link) significance [WIP-DC2005].

There is no question that IPv6 LLU addresses never can be object of any DNS lookup; but could they be the acting subject of DNS resolver and perhaps a DNS server? Could the DNS query and also the response facilitate IPv6 LLU *endpoint* addresses?

This questions will be investigated here and a solution will be outlined using the DNS implementation '**djbdnscurve6**' [14] together with the separate DNS stub resolver library available with '`fehQlibs`' [10]. Both solutions are based on D.J. Bernstein's developments [2] and are particular tailored for IPv6 support, including all required functions to deal with parsing and representation of IPv6 addresses, which are partially taken from Felix von Leitner's patches [18].

### 1.1 IPv6 LLU support and the Interface Index

Unlike IPv4 addresses, IPv6 addresses are 'scoped'. [fig. 1] gives an overview of this situation.

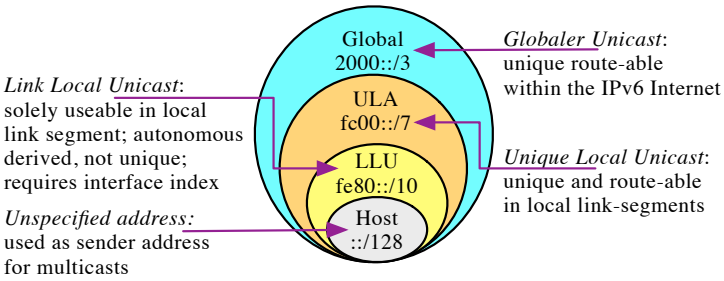


Figure 1: Overview of the IPv6 address hierarchy

In the IPv4 world we consider 'public' and 'private' IP addresses (together with the loopback `127/8` network and the multicast regime). Within IPv6, the situation is more strictly structured using the high order bits of the IPv6 address [8].

IPv6 LLU addresses – like private IPv4 addresses – are not route-able and usable only on the attached local link-segment. Unlike IPv4 private addresses however, each link-segment has its own name space. In practice, this means that the same IPv6 LLU address can be used on different links [31].

In order to facilitate a distinction, the kernel needs to know to which link-segment an IPv6 LLU address belongs to; thus the interface to which it is bound. In a typical case, one may consider a system having an *Ethernet* and a *WiFi* interface with the following addresses:

- `fe80::1%eth0`
- `fe80::2%wifi0`

where the '`%`' is the usual delimiter to separate an IPv6 address from its interface identifier.

The lower 64 bit part of the IPv6 address is typically build from the interface MAC address (or using the 'privacy extension'). Fixed addresses independent from the hardware are common for Internet servers. Often, the none route-able part indicates a certain use, like `2001:1ab::fefe:53` for a DNS service.

In IPv6 the *interface index* (aka the *scope index*) is part of the socket definition. For route-able addresses it is simply '`0`', while for LLU addresses it is derived from the interface. The kernel provides a translation between the interface name (in Unix often `eth0`) and its numeric index, ie. `1`.

This matching is done either statically – upon the boot of the operating system – or dynamically, whenever a interface is added and made available. An interface index may also be removed, upon disabling the interface. This behaviour corresponds with the usage of modern IT equipment – like smartphones – entering and exiting a 'flight mode'.

However, we not only need to consider *physical interfaces* here, but also taking care of additional *virtual interfaces*: The Operating System may create (or dismiss) a virtual or logical interface on demand: At first it becomes assigned an EUI-64 (MAC) address and as second step the corresponding IPv6 unique and multicast addresses are derived algorithmically (SLAAC) [29] [1].

A well-known 'logical' interface is the loopback interface, often named `lo0`. For IPv4, the loopback interface is assigned with the `127/8` net (and not only the IPv4 address `127.0.0.1`). Rather, for IPv6 two differently scoped loopback addresses exist:

- Global scoped: `::1`

- Local scoped: `fe80::1%lo0`

The kernel sees those as different entities and allows binding to each of them.

## 1.2 DNS in the IPv6 working model

In IPv6 networks, the LLU addresses are used to setup a network automatically employing the ICMPv6 protocol:

- *Stateless Address Autoconfiguration* (SLAAC) is used to determine the uniqueness of the autonomous chosen and configured IPv6 address of the interface on a link-segment.
- *Neighbor Discovery Protocol* (NDP) [22] is the substitute for IPv4's ARP (*Address Resolution Protocol*) [23].
- *Neighbor Solicitation* (NS) and *Neighbor Advertisement* (NA) [22] are used to provide a cache for the neighbor's link-addresses.
- *Router Advertisements* (RA) and *Router Solicitation* (RS) together with *Unsolicited Router Advertisements* [22] are able to transmit IPv6 routing prefixes, the set of default routers, the MTU size, and in particular a list of IPv6 addresses of DNS servers to all nodes on the local link-segment [16].

Given the capabilities to provide DNS information in an IPv6 network, DHCPv6 [21] can be used or – as explained – the RA protocol. In the last case, the messages are encapsulated as ICMPv6 packages with the IPv6 LLU address of the sender and targeting either the link-local multicast address, or the LLU address of a specific node on the link-segment.

In short, within IPv6 networks packets with LLU addresses are used to transmit messages containing *network configuration information*. It would be a natural choice to setup a DNS service – either as server or as resolver – to use IPv6 LLU addresses as endpoints for both the query and the reply.

## 1.3 Use cases for DNS IPv6 LLU endpoint addresses

Applying **djbdnscurve6** and applications based on `fehQlibs` with its stub resolver routines, we

1. can setup a DNS (content or cache) server listening on any interface with the provided IPv6 LLU address,
2. can build network applications successfully querying DNS servers given their LLU addresses while being attached to the same local link-segment.

Advantage 1:
On this link-segment DNS query/responses can be operated (irrespectively of the content provided by the Resource Records RR) by the usage of LLU addresses. The transmitted IPv6 data packets are limited to the local link-segment only and are not visible outside (except someone forwards these unsolicited to an outside eavesdropper).

Advantage 2:
Within that link-segment we are under control of our DNS resources. DNS poisoning of the stub resolvers is impossible from sources outside this local net.

Advantage 3:
DNS allows to be setup in a 'split-horizon' manner. The DNS information (in particular hints to the authoritative name servers) can be tailored to the local needs.

Advantage 4:
Location based services (LBS) [28] [27] can be defined and be made *authoritative* in the local net only.

Disadvantage:
A DNS Cache Resolver (like **dnscache**) solely bound to the IPv6 LLU address can not share its cached DNS information with potential DNS clients outside the link-segment.

We will discuss later, how this disadvantage can be compensated for and mitigated by applying a socket binding technique called '*reverse IPv6 anycasting*'.

In summary, using the local link-segment for the communication and DNS message exchange between the DNS stub resolver and a DNS server – in particular a caching Name server – is beneficial and provides anonymity for the stub resolvers while refraining them to use other public DNS caches like Google's '`8.8.8.8`'.

# 2 DNS server with IPv6 LLU support

Within **djbdnscurve6** five DNS server daemons are provided:

- **tinydns** is a UDP-only DNS content server.
- **walldns** is a UDP-only proxy DNS server which might be used to shelter a potential intranet from the Internet while providing synthetic responses to DNS queries.
- **dnscache** is a UDP/TCP DNS full resolver able to validate and cache the responses in memory.
- **rbldns** is an IPv6-enabled Relay Blacklist server.
- **axfrdns** can be used as DNS zone transfer server, typically invoked by **tcpserver** [13] or **sslserver**[12] (in case TLS is an option).

The principal architecture of the DNS servers follow the idea to use IPv6 addresses everywhere. Thus, IPv4 addresses are usually (expect for the 'A' type RR in **tinydns**) considered as *IPv6-mapped IPv4 addresses* [4]. This holds in particular for **dnscache**: Here, the data structure is enhanced (w.r.t. to Daniel J. Bernstein's original work [2]) to use 128 bit addresses potentially together with the required interface indices.

As a side-effect, the DNS resolver routines are capable of handling up to 32 name servers; enough to cover the `a-m.root-servers.net` [15] supporting both their IPv4 and IPv6 addresses.

Common to all servers – including **tcpserver** and **sslserver** – is the capability to bind to any IPv6 address (and of course IPv4 addresses too).

## 2.1 Binding to IPv6 LLU addresses

Though IPv6 is now on the market since 25 years and Google has evaluated that its use is approaching now 30%[1], applying IPv6 LLU addresses for DNS (unicast) services is not well defined yet.

Within **djbdnscurve6** and the auxiliary tools **ucspi-tcp6** [13] and **ucspi-ssl** [12] two different ways to provide the required interface index are used:

a) *Composite IPv6 LLU address* including the interface index followed by the % sign: `fe80::fefe%eth0`.
Here, the symbolic name for the interface is used and not the interface index per se. As already mentioned, the Operating System chooses the enumerated value of the interface index on its own behalf. The routines '`socket_getifidx`' and '`socket_getifname`' are sufficient to do the translation. In this case, a parser is required to cope with this particular address format.

b) *Auxiliary provisioning* of the interface index (name) together with the IP(v6) address as an independent (optional) argument defaulting to '`0`'.

In any case, *compactified* IPv6 can be used as defined in [8]. For ULA and standard IPv6 addresses the interface index is just defined '`0`' and not required to bind to a specific interface.

[1] see: https://www.google.com/intl/en/ipv6/statistics.html

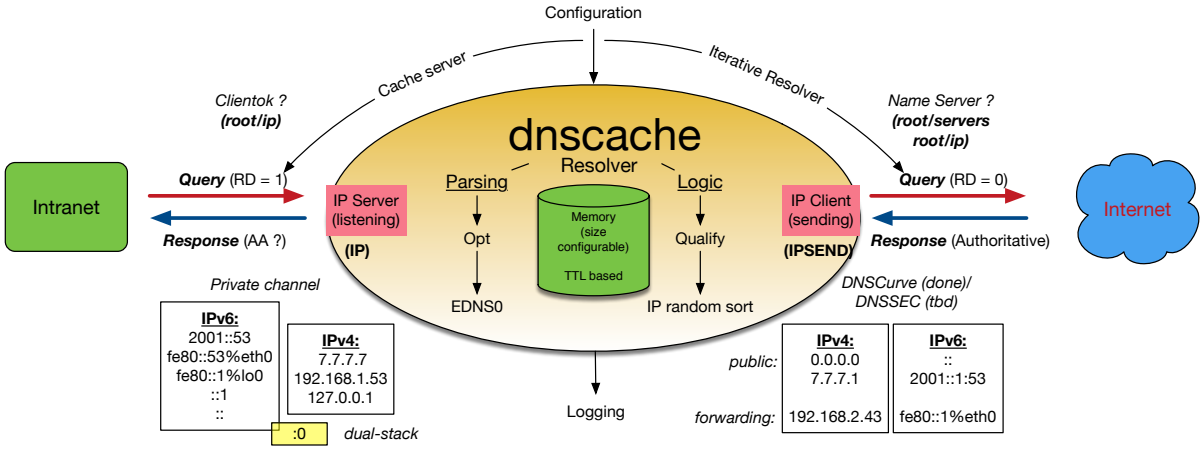


Figure 2: Network setup with **dnscache** using distinct listening and resolving IPs; the notation '`:0`' and '`::`' in the sample in the text below

## 2.2 DNS usage of IPv6 LLU endpoint addresses

For DNS cache servers/full resolvers with enabled IPv6 LLU binding support DNS listening capabilities for queries can be separated from the resolving ones: A DNS cache server (like **dnscache**) can now accept DNS queries from the local link-segment while the actual resolving is realized using its (routeable) IPv4/IPv6 address [fig. 2]. **dnscache** provides the facility to accept queries from authorized sources only, given their client IP address (or client IP net by means of the leading octets).

For any given DNS cache, its server IP address needs to deployed to the clients or even may be public (though a *Open Resolver* is not beneficial and considered to be a risk/bad practice). However, apart from a multi-homing setup of a DNS cache, his feature can be used to support DNS clients on the IPv6 local link-segment.

Advantage 5:
Supporting IPv6 LLU endpoints addresses empowers a DNS cache server to disentangle the queries/replies of the DNS clients from the cache servers name resolution public address improving robustness and resilience.

Advantage 6:
IPv6 LLU addresses come 'for free'. They are SLAAC auto-configured and don't need to be additionally deployed; in contrast to ULA addresses. In particular, they are not subject of IPv6' 'privacy extension'.

## 2.3 Simultaneous binding to IPv4 and IPv6

It might be beneficial, if a daemon has the capability not only to bind to a single IP address of the Operating System, but rather to all. For instance, the Apache web server uses '0' as short-cut argument to bind to all IP addresses.

While a DNS content server – like **tinydns** – allows to use the same DNS zone file for several instances binding to different interfaces or IP addresses; for a caching name server – like **dnscache** – keeping the DNS information in memory, this workaround is not beneficial. Rather, the same cache server instances should

be used to service all interfaces and IP addresses.

We may need to distinguish among the following cases:

1. A legacy binding to all IPv4 addresses.
2. A concurrent binding to both IPv4 and IPv6 addresses (*dual stack*),
3. A dynamic binding to all static and dynamic IPv6 addresses even for virtual interfaces created after the daemon has been started.

In order to facilitate the usage of a common binding to both IPv4 and IPv6 addresses, a 'pseudo' IP address '`:0`' is introduced. If this 'IP address' is given, the server is told to bind to all available IPv4 and IPv6 addresses available on start.

### 2.4 Reverse IPv6 Anycasting

By virtue of the socket routines available in the `fehQlibs`, point three in the previous section can be accomplished with 'reverse IPv6 anycasting support'. IPv6 provides the capability to support this feature using a *socket option*. Unfortunately, this has different notions for Linux/BSD/OmniOS and needs to be accustomed for using a pre-compiler flag [lis. 1].

Here, we need to notice that IPv6 allows the use of the 'unspecified' address [fig. 1]. In case '`::`' is given as binding address, 'reverse IPv6 anycasting' is honoured and DNS queries and responses to dynamically build interfaces are possible even if the DNS server is already running.

Concerning the use cases in section 1.3, a further advantage can now be added:

Advantage 7:
For *Software Defined Network* (SDN) [7] with 'programmed' link-addresses created and dismissed on demand, reverse IPv6 anycasting allows the use of a DNS cache and/or content server in order to provision the required DNS information on demand even on temporary link-segments.

```
int socket_ip6anycast(int s)
{
  int opt = 1;
  int r;

#ifdef GEN_IP_PKTINFO /* Linux */
  r = setsockopt(s,
      IPPROTO_IP,GEN_IP_PKTINFO,
      &opt,sizeof(opt));
#elif IP_PKTINFO  /* Solaris */
  r = setsockopt(s,
      IPPROTO_IP,IP_PKTINFO,
      &opt,sizeof(opt));
#elif IP_RECVDSTADDR  /* BSD */
  r = setsockopt(s,
      IPPROTO_IP,IP_RECVDSTADDR,
      &opt,sizeof(opt));
#elif IPV6_RECVDSTADDR
  if (!ipv4socket)
    setsockopt(s,IPPROTO_IPV6,IPV6_ONLY,
    &opt,sizeof(opt));
  r = setsockopt(s,
      IPPROTO_IPV6,IP_RECVDSTADDR,
      &opt,sizeof(opt));
#endif
  return r;
}
```

Listing 1: Reverse IPv6 anycasting primitive for different Unix OS; `ipv4socket` is a global variable

## 3 DNS (stub) resolvers enhanced for IPv6 LLU addresses

In the previous chapter we discussed the *server/daemon* situation with regard to `*iX` systems. On the *client* side, the situation is however different: The *server* may bind to any IP(v6) address, whereas the *client* must use (one of) the unicast address available on the interface sending a message, in our case, the DNS query.

While all Operating Systems are based on the same IPv4 stack originating from the legacy Unix/BSD systems (*Berkeley sockets*),

concerning IPv6 the situation is the following:

- In the BSD (and in the MacOS Darwin) kernel the results of KAME/WIDE project have been incorporated to provide a solid IPv6 implementation [17], [32].
- In Linux, starting with kernel version 2.4 a preemptive-able network stack was introduced, also supporting IPv6 [25].
- For the Windows operating system, Microsoft has developed IPv6 support starting with Windows XP and employing an own extension of the network interface [19].

In practice, this leads to a recognizable difference in the socket usage, in particular w.r.t to the understanding of an interface, while the main POSIX features defined in [6], [26] are still present.

### 3.1 DNS configuration based on `/etc/resolv.conf`

The classical way for `*iX` Operating Systems is to depend on configuration files like

- `/etc/hosts`
- `/etc/resolv.conf`
- `/etc/nsswitch.conf`

In the advent of the **systemd** [24] subcomponent for Linux this has changed, but is outside the scope of this paper, though **systemd** provides DNS capabilities [fig. 3].

Unfortunately, the scope and usage of `/etc/resolv.conf` is not discussed in any RFC, while at least the original DNS RFC 1034 [20] mentions a configuration file. In fact, according to Wikipedia[2] there is a heritage of the *Berkeley Internet Name Daemon* BIND implementation. Since there is no formal description how this file shall be structured, `/etc/resolv.conf` is practically unusable to host IPv6 (LLU) addresses[3].

It is important to recall that `/etc/resolv.conf` does not necessarily contain *static* information (unlike `/etc/hosts`) but rather is subject to be *rewrit*ten be DHCP(v6) [21] and/or the RA daemon like **radvd**.

The situation to use IPv6 (LLU) addresses within `/etc/hosts` is not clear and depends on the Operating System. In FreeBSD there is a 'bug' which prevents recognizing LLU addresses but rather truncates those and interprets them as hostname.

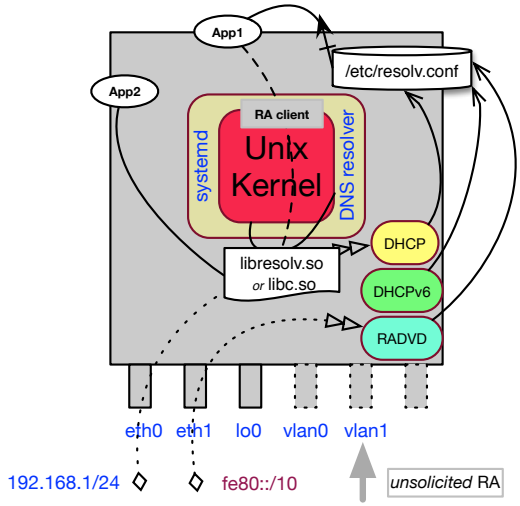


Figure 3: Interaction of the applications with the `*iX` DNS stub resolver libraries. DNS configuration data maybe received via DNS or RA

### 3.2 Alternate DNS configuration of the stub resolver

The DNS stub resolver consists of a library used by the Unix kernel or any user space ap-

[2] https://en.wikipedia.org/wiki/Resolv.conf

[3] https://www.freebsd.org/cgi/man.cgi?query=resolv.conf&sektion=5

plication [fig. 3]. Currently, there is a discussion to use 'DNS-over-HTTP(S)' DoH [9], in particular for Web browsers.

This actually may lead to the following distinctions:

1. The application depends on the core `*iX` DNS Operating System information as provided *centrally* (and potentially updated by DHCP(v6)/RA) as available in `/etc/resolv.conf` ↪ this is the traditional `*iX` way [fig. 3].
2. The Operating System may provide the application a DNS information which is specific for the *interface* the application actually uses ↪ this is the current Windows approach.
3. The application may use an *application specific* tailored DNS configuration it relies upon ↪ DoH and the solution discussed in the next section.

### 3.3 Feeding the stub resolver with DNS hints

A DNS stub resolver needs to be fed with some information in order to start and succeed with the query:

- The *IP addresses* of the (recursive) name server to query, called `RDNSS` in [16].
- The *local domain* added to queries without a full qualified domain name.

Within **djbdnscurve6** and the `fehQlibs` D.J. Bernstein's approach is followed to provision this information as environment variables

- `$DNSCACHEIP`
- `$LOCALDOMAIN`
- `$DNSREWRITE`

to the DNS stub resolver. `$DNSCACHEIP` is a list defining up to 32 (composite) IP addresses (thus potentially including the interface index) of the full resolvers or DNS servers to query. For unqualified DNS queries, the `$LOCALDOMAIN` information can be given additionally; which is equivalent to the *domain* directive in `/etc/resolv.conf`. `$DNSREWRITE` points to a file potentially re-writing a given FQDN[4].

It shall be noted, that these DNS hints are subject of *on-demand change* and in case the DNS stub resolver is capable to do so, are re-evaluated. Thus, dynamical changes of the DNS configuration are recognized and honoured.

### 3.4 Possible solutions to use a DNS stub resolver (library) by an application

In section 3.2 we already discussed the different ways an application depend on the provisioned DNS information. Lets now introduce a model for that, showing two different scenarios.

Scenario 1:
In [fig. 3] the system is equipped with applications making use of the systems `libresolv` library which reads the provisioned IP addresses of DNS full resolvers to contact. However, configuring a DNS stub on `eth1` and the corresponding LLU address given external sources on that link-segment is impossible, since the interface name and the interface index have local meaning only. The same would hold on any DNS resolvers available on a virtual network adaptor, e.g. `vlan0`. This problem is apparent for tunneling interfaces, typically set up by a VPN: DNS traffic is now centralized re-direct to a DNS resolver on a now dynamically re-defined content of `/etc/resolv.conf`.

Scenario 2:
Alternatively, considering the possibility to setup the required DNS information locally

[4] http://cr.yp.to/djbdns/qualify.html

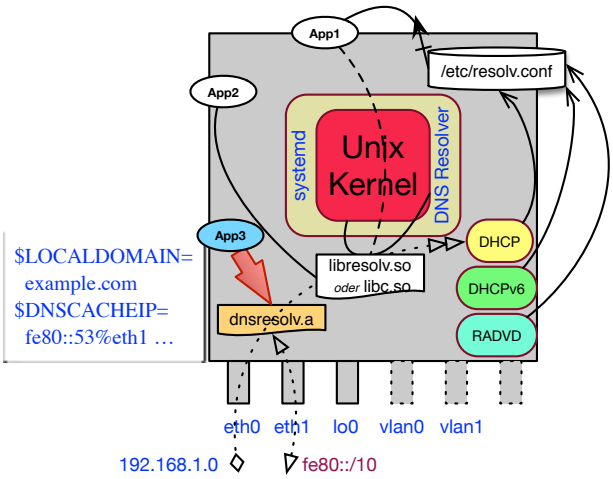


Figure 4: **App3** linked with `dnsresolv` (as DNS stub resolver provider) while using local variables to instruct the DNS resolvers employing the IPv6 LLU address on interface `eth1` for DNS lookup

by means of `dnsresolv` [fig. 4] allows to read those parameters from the user's environment. Application starting from here and linked with `dnsresolv` will catch up those instructions for the DNS stub resolver accordingly. If these directives are missing, a fall-back to the system setting (`/etc/resolv.conf`) is foreseen.

On the bottom line, the user has now the regained capability and perhaps responsibility to care about its own DNS configuration, in case the application permits it. This can be compared with the possibility in the web browser to switch DoH on or off and not just depending on a pre-configured setting which might be infringed with the user's demands.

### 3.5 DNS resolver based on dnsresolv

In order to benefit from that architecture, a potential DNS (stub) client can be build with three options using either:

1. `libdnsresolv.so` – dynamic, position independent (PIC) 'shared object' library.
2. `dnsresolv.a` – static DNS resolver lib (linked to `libdnsresolv.a`).
3. `dnscresolv.a` – static DNS resolver lib including CurveDNS capabilities from **djbdnscurve6**.

Apart from the CurveDNS support within `dnscresolv.a`, these libraries provide the same DNS primitives and a common API; though not compatible with the ISC's BIND[5] ones, which D.J. Bernstein even used within **qmail** [3]. Since low-level IP address parsing and string/byte handling and in particular IPv6 parsing is taken from the `fehQlibs`, they have to be linked in addition.

| Module | **Linux** `AMD64` size [Byte] | **FreeBSD** `ARM64` size [Byte] |
|---|---|---|
| libdnsresolv.so | 43593 | 208632 |
| dnsresolv.a | 71406 | 64254 |
| dnscresolv.a | 81426 | 75882 |
| libqlibs.so | 85713 | 218296 |
| qlibs.a | 150736 | 132350 |

Table 1: Size comparison of the **dns(c)resolv** and `fehQlibs` libraries on Linux (gcc 4.7.2, glibc 2.13) and FreeBSD (clang 6.0.1, ld 6.0.1)

As can be depicted from [tab. 1], the resulting libraries are quite small. This makes them in particular useful for systems with limited CPU power and main memory, which could be a IoT device for instance. In fact, the smallness of those libraries does not put a heavy burden on applications being statically linked with those. Generating none-recursive DNS client modules for **s/qmail** [11] like **dnsip**, **dnsmxip** and others, show up to have a common size of about 52 KByte (under Linux) including `dnsresolv.a` and `qlibs.a` statically

[5] a description of the BIND APIs can be found here: https://bind9.readthedocs.io/en/latest/libdns.html

linked. The inclusion of CurveDNS routines enhance the client routines by about 10 kByte and further 10 kBytes are required for recursive DNS clients like **dnsqr** [14].

## 4 Summary

With **djbdnscurve6** and the DNS stub library provided in `fehQlibs` a solution is given taking care of IPv6 LLU endpoint addresses for DNS queries and responses. However, this does not allow to proliferate IPv6 LLU addresses in DNS as exempted by [5].

The DNS stub resolver can be configured by user-space environment variables. The solution also provides binding to dynamically created IPv6 addresses on ephemeral interfaces generated by software defined link-segments.

The DNS stub resolver – available with and without CurveDNS capabilities – is stable and mature and in particular suited for devices with restricted sources. The code is available (enhancing D.J.Bernstein's library functions) without restriction and well documented.

Though many current requirements for DNS query and response are missing (like DNSSec and multicast support), they are not infringing with the proposed solution.

## References


[1] A. Badach and E. Hoffmann. *Technik der IP-Netze*. 4th Edition. Hanser Verlag, 2019.

[2] D.J. Bernstein. *djbdns*. https://cr.yp.to/djbdns.html. 2019.

[3] D.J. Bernstein. *qmail*. https://cr.yp.to/qmail.html. Jan. 2010. URL: https://cr.yp.to/qmail.html.

[4] M. Blanchet. *Special-Use IPv6 Addresses*. https://tools.ietf.org/html/rfc5156. Apr. 2008.

[5] A. Durand, J. Ihren, and P. Savola. *Operational Considerations and Issues with IPv6 DNS*. https://tools.ietf.org/html/rfc4472. 2006.

[6] R. Gilligan et al. *Basic Socket Interface Extensions for IPv6*. https://tools.ietf.org/html/rfc3493. 2003.

[7] E. Haleplidis and K. Pentikousis. *Software-Defined Networking (SDN): Layers and Architecture Terminology*. https://tools.ietf.org/html/rfc7426. Jan. 2015.

[8] R. Hinden and S. Deering. *IP Version 6 Addressing Architecture*. https://tools.ietf.org/html/rfc4291. 2006.

[9] P. Hoffman and P. McManus. *DNS Queries over HTTPS (DoH)*. https://tools.ietf.org/html/rfc8484. Oct. 2018.

[10] E. Hoffmann. *fehQlibs*. https://www.fehcom.de/ipnet/qlibs.html. 2019.

[11] E. Hoffmann. *s/qmail*. https://www.fehcom.de/sqmail.html. 2020.

[12] E. Hoffmann. *TLS encryption for Client/Server IPv6/IPv4 communication*. https://www.fehcom.de/ipnet/ucspi-ssl.html. Dec. 2019.

[13] E. Hoffmann. *UCSPI for IPv6*. https://www.fehcom.de/ipnet/ucspi-tcp6.html. Dec. 2019.

[14] E. Hoffmann. *Unified IPv6 DNS Security*. https://www.fehcom.de/ipnet/djbdnscurve6.html. 2019.

[15] Internet Assigned Numer Authority. *Root Servers*. https://www.iana.org/domains/root/servers. Jan. 2020.

[16] J. Jeong et al. *IPv6 Router Advertisement Options for DNS Configuration*. https://tools.ietf.org/html/rfc6106. 2010.

[17] KAME. *The KAME project*. http://www.kame.net/. 1998.

[18] F. von Leitner. *What is djbdns and why does it need IPv6?* https://www.fefe.de/dns/. 2019.

[19] Microsoft. *IPv6*. http://technet.microsoft.com/en-us/network/bb530961. 2012.

[20] P. Mockapetris. *DOMAIN NAMES - CONCEPTS AND FACILITIES*. https://tools.ietf.org/html/rfc1034. Nov. 1987.

[21] T. Mrugalski et al. *Dynamic Host Configuration Protocol for IPv6 (DHCPv6)*. https://tools.ietf.org/html/rfc8415. Nov. 2018.

[22] T. Narten et al. *Neighbor Discovery for IP version 6 (IPv6)*. http://www.ietf.org/rfc/rfc4861. 2007.

[23] Plummer, D. C. *An Ethernet Address Resolution Protocol*. https://tools.ietf.org/html/rfc826. Nov. 1982.

[24] L. Poettering. *systemd*. https://github.com/systemd/systemd. Jan. 2020.

[25] USAGI Project. *Linux IPv6 Development Project*. http://www.linux-ipv6.org/. 2006.

[26] W. Stevens, M. Thomas, and T. Jinmei. *Advanced Sockets Application Program Interface (API) for IPv6*. https://tools.ietf.org/html/rfc3542. May 2003.

[27] M. Thomson and R. Bellis. *Location Information Server (LIS) Discovery / Using IP Addresses and Reverse DNS*. https://tools.ietf.org/html/rfc8484. Apr. 2014.

[28] M. Thomson and Winterbottom J. *Discovering the Local Location Information Server (LIS)*. http://www.rfc-editor.org/rfc/rfc5986. Sept. 2010.

[29] S. Thomson, T. Narten, and T. Jinmei. *IPv6 Stateless Address Autoconfiguration*. http://www.ietf.org/rfc/rfc4862. 2007.

[30] S. Thomson et al. *DNS Extensions to Support IP Version 6*. https://tools.ietf.org/html/rfc3596. 2003.

[31] M. Wasserman and P. Seite. *Current Practices for Multiple-Interface Hosts*. https://tools.ietf.org/html/rfc6419. 2011.

[32] WIDE. *WIDE v6 working group*. http://www.v6.wide.ad.jp/. 2003.