FOSDEM 2020

Bruxelles

IPv6 LLU Endpoint Support in DNS

.. and its implementation in djbdnscurve6

Erwin Hoffmann

feh@fehcom.de

https://www.fehcom.de

(February 2, 2020)

Outline

djbdnscurve6 is a fork of Daniel Bernstein's **djbdns** with focus on complete user space IPv6 support. It is based on the **fehQlibs** library providing the required 'C' routines for IP address parsing, socket calls together with byte & string handling. The **fehQlibs** also include an DNS stub resolver library. Using these libraries, DNS servers and DNS resolvers can efficiently use IPv6 LLU endpoint addresses for DNS message exchange.

Topics:

1. Short history and coverage of **fehQlibs** and **djbdnscurve6**
2. Benefits of using IPv6 LLU endpoint addresses for DNS services
3. Applying IPv6 LLU support for servers
4. Integration of IPv6 LLU endpoint addresses for DNS stub resolvers and application using those
5. Use cases and outlook

The achieved results are partially based on my lectures 'Moderne Netzstrukturen' given at the *Frankfurt University of Applied Sciences* and 'Distributed Systems' given at the *Vietnamese German University* in Hoh-Chi-Minh City while applying those to DJB's routines and enhancing them for missing functionality, like IPv6 support.

Some more details about the IPv6 protocol can be found in my book '*Technik der IP-Netze*' (German only).

History and coverage of **fehQlibs** and **djbdnscurve6**

- Since now 20 years I try to keep the SW of *Daniel J. Berstein* (DJB) up-to-date; among others, like *Felix von Leitner* (fefe) [6].
- Apart from **qmail**, for which I have created a fork **s/qmail**, over the last two years I published on collaboration with *Kai Peter* the so-called **qlibs** or **fehQlibs** [5].
- Applying that library, the existing DNS implementions of DJB – **djbdns** – has been refactored entirely, including native IPv6 support and also providing an integrated implementation of CurveDNS for the **dnscache** server [4].
- Current release is **djbdnscurve6-36a** together with **fehQlibs-12c**.

Server	TCP	UDP	EDNS0	CurveDNS
tinydns	-	✓	-	(Vers 3)
rbldns	-	✓	-	n/a
walldns	-	✓	-	-
dnscache	✓	✓	✓ (resolving)	✓
axfrdns	✓	n/a	n/a	n/a

Table: DNS server modules in * and their capabilities (n/a: not applicable)

Client	TCP	UDP	EDNS0	CurveDNS
dnsip	-	✓	-	✓
dnsmx	-	✓	-	✓
dnsname	-	✓	-	✓
dnstxt	-	✓	-	✓

Table: DNS client modules in **djbdnscurve6** and their capabilities

↪ Goal was an implementation of DNS caching server, dealing the well known problem of the '*Byzantinean Generals*' [3] for 'Distributed Systems' without using digital signatures as available in DNSSec.

Why IPv6 LLU support for DNS?

(raison d'être)

The use of IPv6 Link Local Unicast (LLU) addresses in the context of DNS is not very clear described. Here, we have to distinguish two general cases:

Case 1: IPv6 LLU addresses in Zone files

> *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].***

RFC 4471 [11] simply expresses the impossibility to provide successfully limited-scoped IPv6 addresses outside the link-local segment.

Case 2: IPv6 LLU endpoint addresses

Nobody forbids us from using IPv6 LLU addresses to be the endpoint of a DNS service. However, we have to solve two distinct problems:

1. The DNS server must be able to bind to an IPv6 LLU address; thus posses knowledge of the respective *Interface Index*: `fe80::53%eth0`.
2. The DNS (stub) resolver must be supplied with a hint via which interface a DNS server is reachable given its IPv6 LLU address.

↪ Solutions for these challenges are given in this talk using the particular DNS implementation of **djbdnscurve6** as a blueprint; but not requiring those.

IPv6 SLAAC and Router Advertisements

After a successful *Stateless Address Autoconfiguration* (SLAAC) of the IPv6 node, *Router Advertisements* (RA) are used to provision the nodes with configuration information. In particular, the IPv6 routers are reachable on the local link segment via IPv6 LLU addresses.

RFC 8106 [15] defines the RA option 25 allowing to deploy

- a list of *Recursive DNS Servers* `RDNSS` given their IPv6 address and
- a *DNS Search List* `DNSSL`.

RFC 8106 does not state, which kind of IPv6 address shall be used here. The client receiving this information may associate the IPv6 LLU address with the *link-local segment* at which the ICMPv6 message was received.

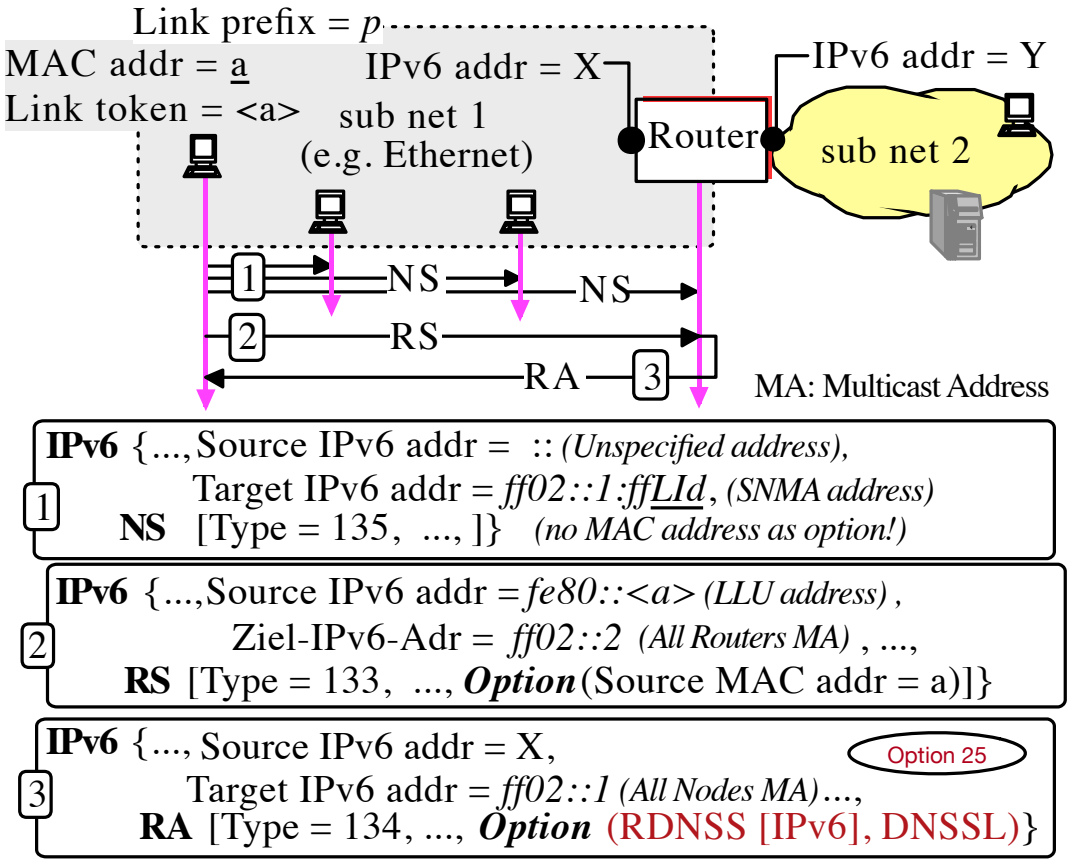

Figure: Principal of IPv6 SLAAC and subsequent provisioning of network configuration by means of router advertisements (RA); including DNS information

Using the scoped IPv6 address for socket binding

IPv6 carries the idea to have *scoped* addresses. For TCP/UDP socket communication, this requires to add an *Interface Index* to the socket call; defaulting to '0' except for IPv6 *link local unicast* (LLU) addresses telling to which interface to bind with.

↪ IPv6 LLU addresses require an additionally supplied *Interface Index* or *Interface Name* for a successful binding: `fe80::53%eth0`

IPv6 has a hierarchical understanding of the purpose of an IPv6 address given the first bits in here:

- ↓ *Multicast addresses* and in particular the automatic Solicited Node Multicast Address (SNMA).
- ↕ *Unicast addresses* (with scope global, site- and link-local).
- ↑ *Unspecified address* (without a particular scope).

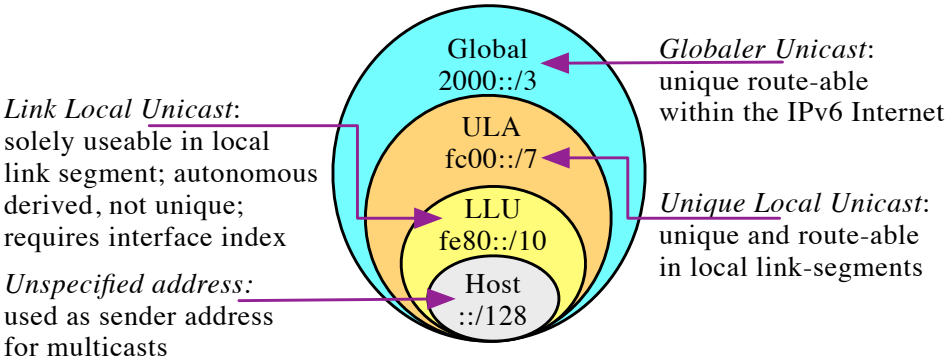

Figure: Overview of the IPv6 address hierarchy

Type	**Net ID**	**Prefix** length
Multicast (MC)	`ff`	/8
Solicited-Node MC	`ff02::1:ff`	/104
All-Node MC	`ff02::1`	/128
All-Router MC	`ff01::2`	/128
mDNSv6 MC [RFC 6762]	`ff01::fb`	/128
Site-local Unicast **SLU**	`fec0`	/10
Link-Local Unicast **LLU**	`fe80`	/10
Unique-Local Unicast **ULA**	`fc00`	/7
IPv4-mapped IPv6	`::ffff`	/96
Loopback	`::1`	/128
Unspecified	`::`	/128

Table: Systematic of IPv6 addresses [10]

Interface Index & Dual Stack binding

Interface Index and binding:

For any servers, in particular DNS servers, we can realize binding to IPv6 LLU addresses by two different schemes:

1. We may include the Interface name as additional argument together with the IPv6 address upon call:
   `tcpserver -Ieth0 fe80::1`
2. We could use a *composite IPv6 address* including both the address and the *Interface Name* linked with the usual '%' (percent) sign (similar to a prefix notation):
   `fe80::53%eth0`

↪ The kernel requires to bind to the interface given its *Interface Index* (aka *scope index*). We can use the IPv6 socket function '`socket_getifidx`' in order to derive this from the *Interface Name*.

Dual Stack binding:

For IPv4, there exist the convenient notation to specify a '0' in order to bind to all available IPv4 addresses upon call.

It is desirable for DNS servers to bind commonly to IPv4 and IPv6 addresses in order to supply the identical information to any clients asking the server, irrespectively if the query arrives via IPv4 or IPv6.

↪ In **djbdnscurve6** and **ucspi-tcp6** as well as **ucspi-ssl**, I've chosen the abbreviation '`:0`' to provide native dual-stack binding. Some care needs to be taken in order to set the correct socket options for the target OS allowing this.

Loopback interfaces:

- IPv4: `127.0.0.1`
- IPv6: `::1` (global scoped)
- IPv6: `fe80::1%lo0` (local scoped)

Reverse IPv6 Anycasting

One particular merit of IPv6 networks is to allow 'dynamic' binding to IPv6 addresses:

> While the daemon is running, it is possible to service newly defined IPv6 addresses and interfaces. I've chosen to use the unspecified IPv6 address '::' to support what is called *reverse IPv6 anycasting*.

Sample with **dnscache**:

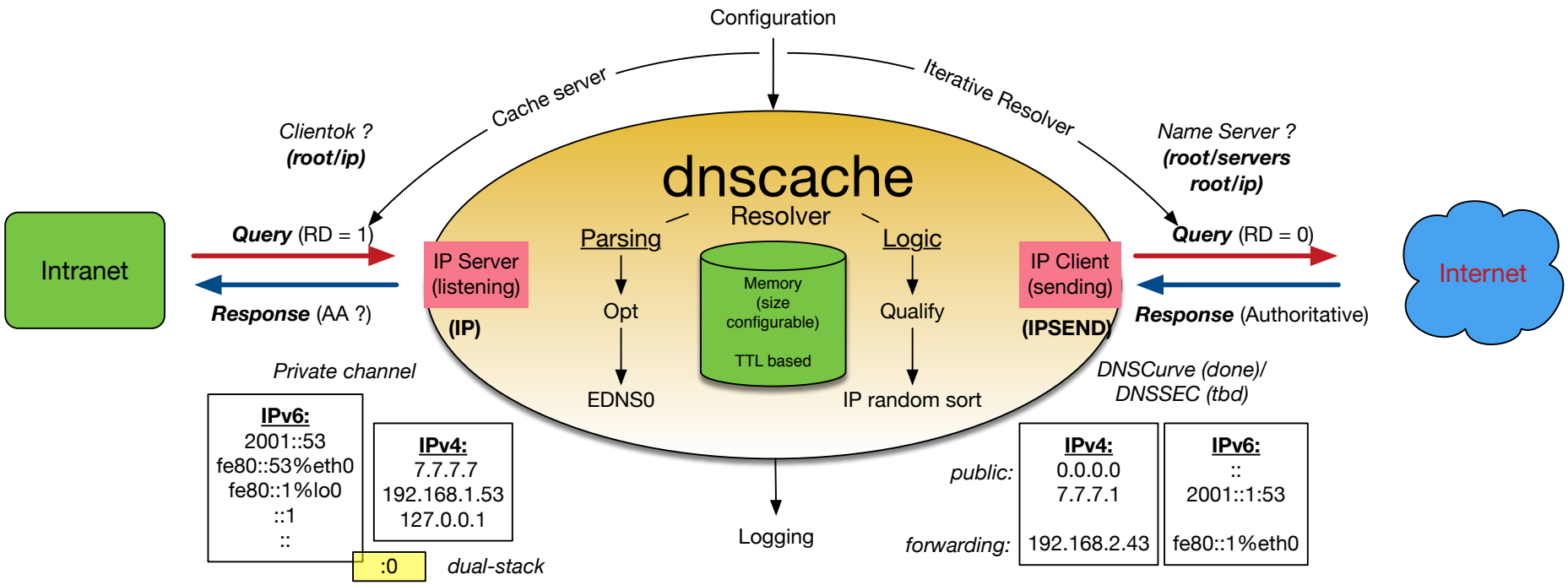

Figure: Network setup with **dnscache** using distinct listening and resolving IPs; using the notation ':0' and '::' for binding

DNS stub resolver and IPv6 LLU endpoints

A DNS (stub) resolver needs to know, which (recursive) resolver to contact and where to store the information. RFC 8106 [15] simply states for DNS *Router Advertisements*:

> *Resolver Repository: Configuration repository with RDNSS addresses and a DNS Search List that a DNS resolver on the host uses for DNS name resolution; for example, the Unix resolver file (i.e., /etc/ resolv.conf) and Windows registry.*

Given the current implementation to store DNS configuration, we may recognize:

1. The legacy Unix way is to store DNS configuration data **system-wide** the file `/etc/resolv.conf`.
2. Since the Windows operating system facilitates a *Registry*, here DNS configuration is attached **interface-specific** considering the interface over which the information is received.
3. The current discussion – in particular the DoH implementation using a crafted URL [16] – associates the DNS configuration **application-specific**.

↪ *Thus, there is no canonical way to store DNS configuration persistently for the stub resolver.*

Daniel Bernstein preferred an environment specific DNS setup, with fall-back to the system-wide `/etc/resolv.conf`. This path is followed here given the implementation of the DNS stub resolver.

DNS (stub) resolver's configuration

Apart from DoH, the central DNS repository is `/etc/resolv.conf` [Fig. 4]. Given its tasks, some deficiencies may be encountered:

- `/etc/resolv.conf` is configured by several competing task.
- `/etc/resolv.conf` uses a none-standardized format.
- Reading `/etc/resolv.conf` needs to implement the 'least common denominator'; parsing & error handling is up to the client.
- Configuration data deployed here by nasty programs could invalidate its content.
- `/etc/resolv.conf` has no understanding of IPv6 LLU addresses.

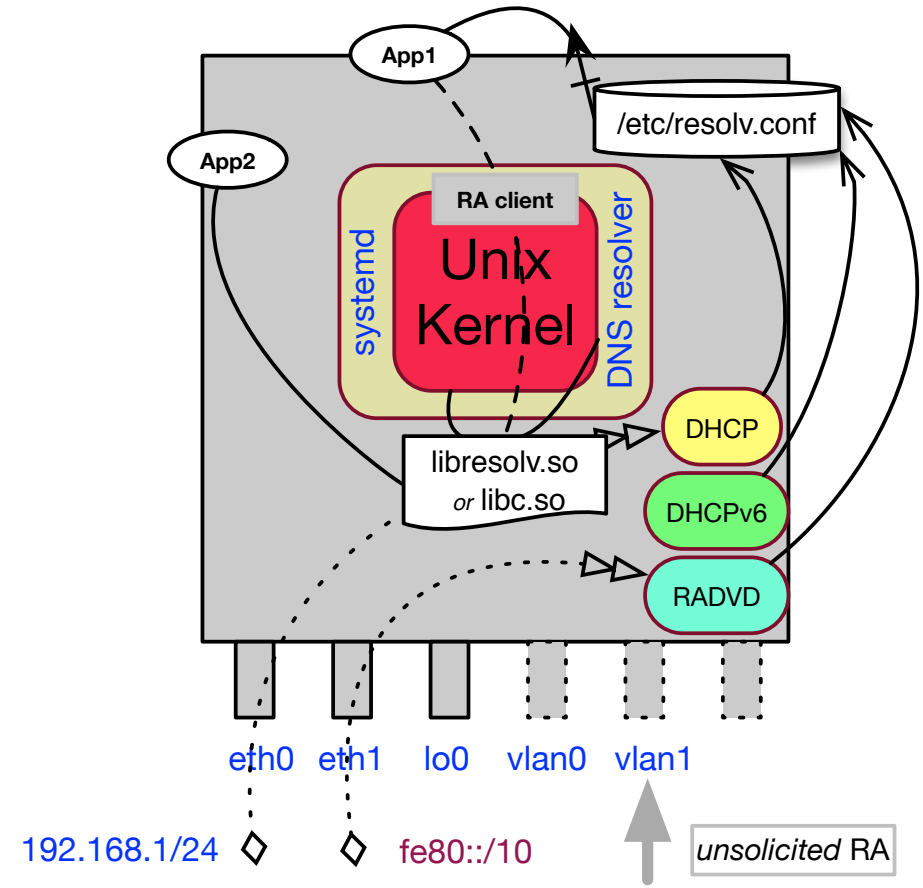

Figure: Usage and provisioning of /etc/resolv.conf

The DNS hint-reading behaviour is based on the standard BIND APIs like '`gethostbyname`' and depending on an (application specific) DNS stub resolver and potentially using the functions available in `libresolv.so`.

Application specific DNS (stub) resolver's setting

Applications linked with `dnscresolv.a` from `djbdnscurve6` or with `dnsresolv.a`/`libdnsresolv.so` from `fehQlibs` [Fig. 5] may evaluate DNS resolver hints given by environment variables pointing to:

- `$DNSCACHEIP=feh80::53%eth1`
- `$LOCALDOMAIN=example.com`

Unlike the original implementation of DJB `$DNSCACHEIP` may be constructed as list of IP addresses:

- IPv6 addresses are given in their compactified format.
- IPv6 LLU addresses needed to be appended with the (local) *Interface Name* in the canonical format including the '%' sign as delimiter.

↪ In case no environment variables are provided, the DNS hints are taken from `/etc/resolv.conf`, while a 'mixed' use is possible.

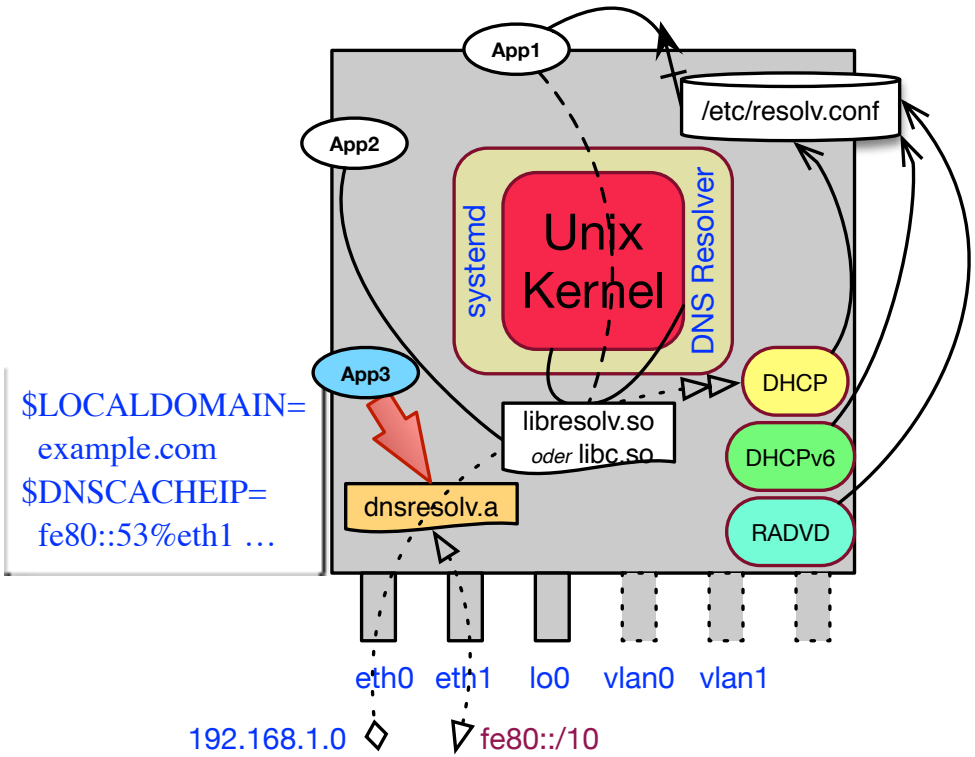

Figure: App3 uses environment provisioned DNS hints together with `dnsresolv.a`

The DNS library `dns[c]resolv.a` is typically statically linked by the application together with `qlibs.a` for IPv4/IPv6 parsing/socket calls and others.

Case Study 1: IoT communication over IPv6 using **dnsresolv**

Item	**Description**
Situation	IoT network provisioned with data via RA & DHCPv6.
Configuration	The IoT network is dynamically auto-configured.
Result	Network infrastructure is 'concealed' and reachable only via IPv6 LLU addresses.
DNS service	Nodes/applications are resolvable by their name applying the IPv6 LLU address of the DNS server (content or cache) for query.
Communication	Internet network communication is facilitated by IPv6 ULA addresses.
Security	▸ Assuming, that no node/application was 'hijacked' and the network is not 'tapped', it can be considered as *Security Compartment*. ▸ If no gateway/NAT device is present, no IPv6 packets are leaked. ▸ Even without explicit data/transport layer encryption, data are confined in the network.
Performance	Refraining from encryption and the required TLS handshake, network performance may reach the maximal capabilities of the nodes/applications.

Table: Case study 1: IoT communication based on IPv6 with applications using `dnsresolv.a`

Case Study 2: Software Defined Networks with virtual Interfaces

Item	Description
Situation	Temporary applications are deployed via *Docker*.
Configuration	Applications and switches are provisioned with VLANs using virtual (temporary) interfaces.
Result	Network infrastructure is 'concealed' and reachable only via IPv6 LLU addresses.
DNS service	▸ DNS requests are realized over the virtual interface: → ***link-scoped***. ▸ DNS cache servers starting 'cold' and don't posses any knowledge of the existing topology.
Communication	▸ The entire network is again 'concealed': → Application ≡ VLAN. ▸ Reachability of services is restricted using the virtual interface only.
Serviceability	▸ Long haul services provisioned with additional VLANs/interfaces need to catch up changing topologies requiring to support reverse IPv6-Anycasts. ▸ Applications using `fehQlibs` can realize this task binding to '::'.

Table: Case study 2: Software Defined Network communication & DNS

Case Study 3: Information Centric Networking (ICN)

Item	Description
Situation	Entering a *realm* the provided services shall be deployed by local (= none-public) URIs: → *Location-to-Service Translation (LoST)*.
DNS	▸ The application (typically a web browser) receives a hint pointing to the local DNS server: → *application-scoped DNS*. ▸ The application receives a *search path* for the local domain: `$LOCALDOMAIN`.
Results	▸ Any realm-specific informations are *application-scoped*; while the 'remaining' device is connected to the usual Internet (perhaps given its *Home-Zone*) and including its public IP address. ▸ Monopolistic information (typically associated with *Google*) provisioning can be avoided; local and specific information become precedence.

Table: Case study 3: Information centric networking & DNS

Outlook & Questions

djbdnscurve6 and the **fehQlibs** provide both a solid platform for DNS and network communication; in particular considering IPv6. My next development steps are:

Network communication:

- Add *Multicast* support with `fehQlibs`.
- Add TCP support for **tinydns**.

DNS support:

- EDNS0 for clients & servers.
- Include CurveDNS natively for **tinydns** → **tinycdns** (**djbdnscurve6** V3).
- Add DNSSec support for **tinydns**.
- Add DNSSec validation for **dnscache**.

Outlook & Questions

djbdnscurve6 and the **fehQlibs** provide both a solid platform for DNS and network communication; in particular considering IPv6. My next development steps are:

Network communication:

- Add *Multicast* support with `fehQlibs`.
- Add TCP support for **tinydns**.

DNS support:

- EDNS0 for clients & servers.
- Include CurveDNS natively for **tinydns** → **tinycdns** (**djbdnscurve6** V3).
- Add DNSSec support for **tinydns**.
- Add DNSSec validation for **dnscache**.

Questions?

Code sizes of djbdnscurve modules

Client	Module size	Server	Module size
dnsip	65824 byte	tinydns (only UDP)	56512 byte
dnsmx	64624 byte	dnscache (+ NaCl)	142552 byte
dnstxt	64624 byte	rbldns (only UDP)	52416 byte

(without NACl: 89304 byte)

Table: Code sizes of djbdnscurve6-35 client- and server modules

Code size: dnsresolv

```
-rwxr-xr-x  1 djbdns  users  39974 Dec  9 15:21 libdnsresolv.so <= -fPIC -shared
-rw-r--r--  1 djbdns  users  65416 Dec  9 15:21 dnsresolv.a     <= -fPIC

-rw-r--r--  1 djbdns  users  79242 Sep 19 23:49 dnscresolv.a    <= with NaCl
```

size dnsresolv.a ↪ https://www.fehcom.de/ipnet/fehQlibs/doxygen/files.html

text	data	bss	dec	hex	filename
958	0	0	958	0x3be	dns_domain.o (ex dnsresolv.a)
647	0	0	647	0x287	dns_dfd.o (ex dnsresolv.a)
358	0	0	358	0x166	dns_dtda.o (ex dnsresolv.a)
1983	0	8	1991	0x7c7	dns_ip.o (ex dnsresolv.a)
1371	0	48	1419	0x58b	dns_ipq.o (ex dnsresolv.a)
707	0	8	715	0x2cb	dns_mx.o (ex dnsresolv.a)
1043	0	8	1051	0x41b	dns_name.o (ex dnsresolv.a)
440	0	0	440	0x1b8	dns_nd.o (ex dnsresolv.a)
581	0	0	581	0x245	dns_packet.o (ex dnsresolv.a)
1176	0	224	1400	0x578	dns_random.o (ex dnsresolv.a)
1271	0	720	1991	0x7c7	dns_rcip.o (ex dnsresolv.a)
1655	0	80	1735	0x6c7	dns_rcrw.o (ex dnsresolv.a)
333	0	112	445	0x1bd	dns_resolve.o (ex dnsresolv.a)
787	0	0	787	0x313	dns_sortip.o (ex dnsresolv.a)
5366	0	0	5366	0x14f6	dns_transmit.o (ex dnsresolv.a)
757	0	8	765	0x2fd	dns_txt.o (ex dnsresolv.a)

size dnscresolv.a ↪ https://www.fehcom.de/ipnet/djbdnscurve6/doxygen/

5048	0	0	5048	0x13b8	dns_transmit.o (ex dnscresolv.a)
4049	0	8	4057	0xfd9	curvedns.o (ex dnscresolv.a)
916	0	0	916	0x394	base32.o (ex dnscresolv.a)
299	0	156	455	0x1c7	serverok.o (ex dnscresolv.a)

Code size: libresolv.a

For Linux (Debian Wheezy):

```
-rw-r--r-- 1 root root  80712 Jun 19  2017 libresolv-2.13.so
-rw-r--r-- 1 root root 126090 Jun 19  2017 libresolv.a
```

↪ roughly twice the size compared to `dnsresolv`

size `libresolv.a`

text	data	bss	dec	hex	filename
6762	0	8848	15610	3cfa	gethnamaddr.o (ex libresolv.a)
805	0	0	805	325	res_comp.o (ex libresolv.a)
11979	4	324	12307	3013	res_debug.o (ex libresolv.a)
1812	128	1025	2965	b95	res_data.o (ex libresolv.a)
838	0	0	838	346	res_mkquery.o (ex libresolv.a)
4624	0	0	4624	1210	res_query.o (ex libresolv.a)
8279	0	4	8283	205b	res_send.o (ex libresolv.a)
497	0	0	497	1f1	inet_net_ntop.o (ex libresolv.a)
1293	0	0	1293	50d	inet_net_pton.o (ex libresolv.a)
269	0	0	269	10d	inet_neta.o (ex libresolv.a)
1246	0	0	1246	4de	base64.o (ex libresolv.a)
1548	0	0	1548	60c	ns_parse.o (ex libresolv.a)
5628	0	0	5628	15fc	ns_name.o (ex libresolv.a)
163	0	0	163	a3	ns_netint.o (ex libresolv.a)
1565	0	0	1565	61d	ns_ttl.o (ex libresolv.a)
9826	0	0	9826	2662	ns_print.o (ex libresolv.a)
1060	0	0	1060	424	ns_samedomain.o (ex libresolv.a)
905	0	0	905	389	ns_date.o (ex libresolv.a)

Sources


[1] *djbdns* `https://cr.yp.to/djbdns.html`


[2] *DNSCurve* `http://dnscurve.org`


[3] Lamport, Shostak, Pease The Byzantine Generals Problem


[4] *djbdnscurve6* `https://www.fehcom.de/ipnet/djbdnscurve6`


[5] *DJBware* `https://www.fehcom.de/djbware.html`


[6] *fefe* `https://www.fefe.de`


[7] *ISC* `https://www.isc.org/community/rfcs/dns/`


[8] *DOMAIN NAMES - CONCEPTS AND FACILITIES* RFC1034


[9] *Requirements for Internet Hosts – Application and Support* RFC1123


[10] *IP Version 6 Addressing Architecture* RFC4291


[11] *Operational Considerations and Issues with IPv6 DNS* RFC4471


[12] *Current Practices for Multiple-Interface Hosts* RFC6491


[13] *Multicast DNS* RFC6762


[14] *DNS Terminology* RFC7719


[15] *IPv6 Router Advertisement Options for DNS Configuration* RC8106


[16] *DNS Queries over HTTPS (DoH)* RFC8484