# GUUG Frühjahrsfachgespräch 2026

# Der DNS-Stubresolver – das unbekannte Wesen

Erwin Hoffmann

[feh@fehcom.de]

5. März 2026

# Überblick

Wir wollen einen Blick in den Maschinenraum von DNS auf unserem Unix-Rechner werfen: Was läuft dort wie und mit wem ab, damit typischerweise für einen nachgefragten Hostname (vulgo: FQDN; also z.B. `www.example.com`) eine IP-Adresse zurück gegeben wird, mit dessen Hilfe ein Kommunikationssocket erstellt werden kann.
Die Details hierzu sind überraschend und unterscheiden sich zudem von Unix zu Linux – und werden sich zudem in der Zukunft noch weiter entwickeln. Hierzu will ich abschliessend einen Ausblick geben.

Programm für Heute:

- Die Komponenten des DNS Stub-Resolvers und Integration in Unix/Linux.
- Die verschiedenen Schritte rund um die Name Resolution.
- Das Datei-Inventar, auf den sich der Stub-Resolver stützt.
- Die Akteure und Mitstreiter beim Einsatz des Stub-Resolvers.
- Die DNS-Alternativen, jenseits der Default-Libraries für Unix/Linux.

# Internet-Applikationen und der Stub-Resolver

Für eine Applikation '**App**' sieht die Nutzung des Stub-Resolvers folgendermassen aus:

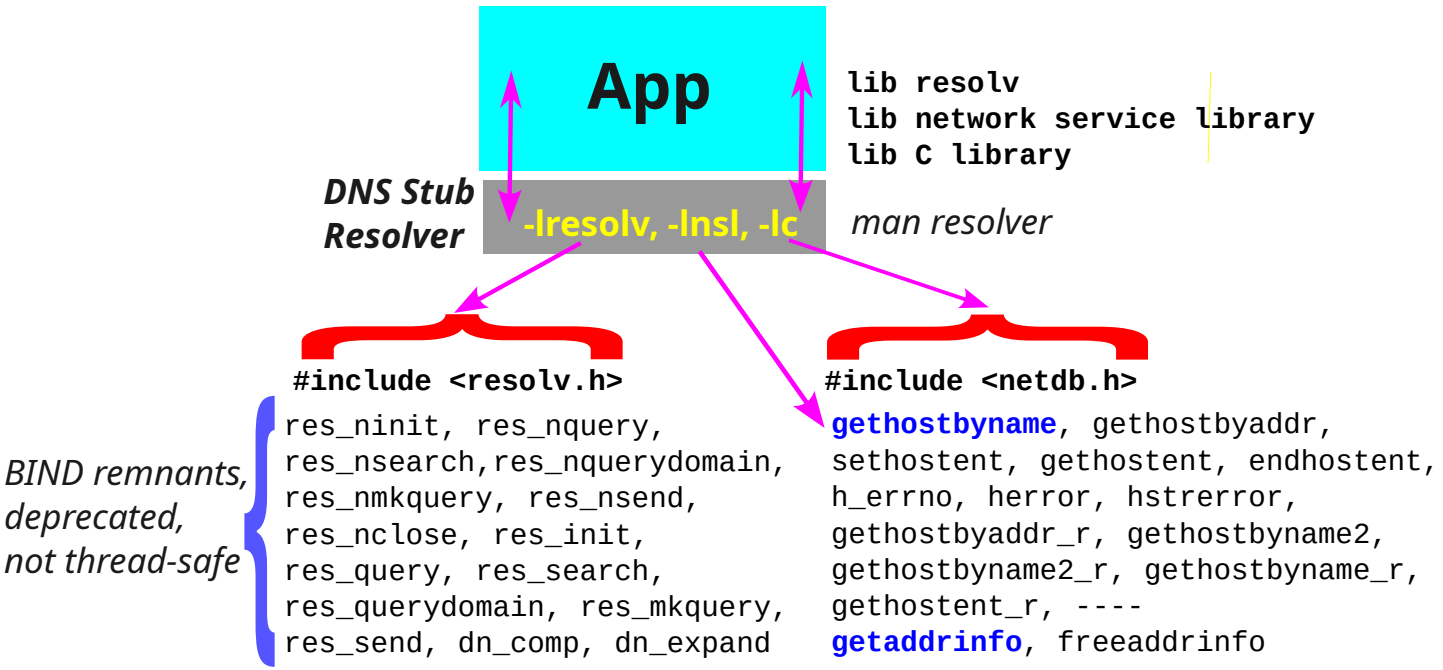



**Abbildung:** Der Stub-Resolver unter Unix und seine Abhängigkeiten

Erkenntnisse:

- Ein Teil des Stub-Resolvers stellt die **libC** bereit.
- (Ältere) Teile des Stub-Resolvers werden über die **libresolv** verfügbar gemacht. Diese erfüllen Funktionen eines 'eigentlichen' DNS Services entsprechend BIND.
- Bei einigen Unixen bietet die *Network Service Library* (**lnsl**) bzw. *Network Service Switch* (**lnss**) ergänzende Stub-Resolver Funktionen als Teil der *Network Information Services* (**NIS**) an.

↪ Dies wird bei den unterschiedlichen Unixen auch unterschiedlich gelöst!

# Was macht der Stub-Resolver und was ist da drin?

Wir schauen uns mal die **libresolv** Library unter einem aktuellen (Debian) Linux an:

```
$ ls -lh /usr/lib/x86_64-linux-gnu/libresolv.a
  -rw-r--r-- 1 root root 82K Dec 28 16:32 /usr/lib/x86_64-linux-gnu/libresolv.a

$ size /usr/lib/x86_64-linux-gnu/libresolv.a
   text    data     bss     dec     hex filename
   1196       0       0    1196     4ac base64.o (ex libresolv.a)
      0       0       0       0       0 compat-gethnamaddr.o (ex libresolv.a)
      0       0       0       0       0 compat-hooks.o (ex libresolv.a)
    605       0       0     605     25d inet_net_ntop.o (ex libresolv.a)
   1446       0       0    1446     5a6 inet_net_pton.o (ex libresolv.a)
    356       0       0     356     164 inet_neta.o (ex libresolv.a)
   1048       0       0    1048     418 ns_date.o (ex libresolv.a)
    520       0       0     520     208 ns_name.o (ex libresolv.a)
    157       0       0     157      9d ns_netint.o (ex libresolv.a)
   1458       0       0    1458     5b2 ns_parse.o (ex libresolv.a)
  11690       0       0   11690    2daa ns_print.o (ex libresolv.a)
    673       0       0     673     2a1 ns_samedomain.o (ex libresolv.a)
   1514       0       0    1514     5ea ns_ttl.o (ex libresolv.a)
    174       0       0     174      ae res-putget.o (ex libresolv.a)
     79       0       0      79      4f res_data.o (ex libresolv.a)
  11529    1932     292   13753    35b9 res_debug.o (ex libresolv.a)
    415       0    1025    1440     5a0 res_hostalias.o (ex libresolv.a)
    377       0       0     377     179 res_isourserver.o (ex libresolv.a)
    157       0       0     157      9d resolv-deprecated.o (ex libresolv.a)
```

**Listing:** Ermittlung der Module der libresolv.a unter Linux 6.12.57+deb13-amd64

↪ Wenig überraschend finden sich hierin auch Routinen zur Verarbeitung von IP-Adressen (**inet_netntop.o**). Aber was soll **resolv-deprecated.o** sein?

# man: gethostbyname

```
NAME
       gethostbyname,  gethostbyaddr, sethostent, gethostent, endhostent, h_er-
       rno,  herror,  hstrerror,  gethostbyaddr_r,  gethostbyname2,  gethostby-
       name2_r, gethostbyname_r, gethostent_r - get network host entry
LIBRARY
       Standard C library (libc, -lc)
SYNOPSIS
       #include <netdb.h>

       void sethostent(int stayopen);
       void endhostent(void);
       [[deprecated]] extern int h_errno;
       [[deprecated]] struct hostent *gethostbyname(const char *name);
       [[deprecated]] struct hostent *gethostbyaddr(const void addr[.len],
                                                    socklen_t len, int type);
       [[deprecated]] void herror(const char *s);
       [[deprecated]] const char *hstrerror(int err);
       /* System V/POSIX extension */
       struct hostent *gethostent(void);
       /* GNU extensions */
       [[deprecated]]
       struct hostent *gethostbyname2(const char *name, int af);
       int gethostent_r(struct hostent *restrict ret,
                        char buf[restrict .buflen], size_t buflen,
                        struct hostent **restrict result,
                        int *restrict h_errnop);
       [[deprecated]]
       int gethostbyaddr_r(const void addr[restrict .len], socklen_t len,
                        int type,
                        struct hostent *restrict ret,
                        char buf[restrict .buflen], size_t buflen,
                        struct hostent **restrict result,
                        int *restrict h_errnop);
       [[deprecated]]
       int gethostbyname_r(const char *restrict name,
                        struct hostent *restrict ret,
                        char buf[restrict .buflen], size_t buflen,
                        struct hostent **restrict result,
                        int *restrict h_errnop);
```

# man: resolver

```
NAME
       res_ninit,  res_nquery,  res_nsearch,  res_nquerydomain,  res_nmkquery,
       res_nsend, res_nclose, res_init, res_query, res_search, res_querydomain,
       res_mkquery, res_send, dn_comp, dn_expand - resolver routines

LIBRARY
       Resolver library (libresolv, -lresolv)

....

DESCRIPTION
       Note: This page is incomplete (various resolver  functions  provided  by
       glibc are not described) and likely out of date.

       The  functions  described  below  make  queries to and interpret the re-
       sponses from Internet domain name servers.
```

# gethostbyname/getaddrinfo in Applikationen

Das Einbinden von gethostbyname() aus den Libraries kann nur dann nachvollzogen werden, falls die zugehörigen Symbole im *Executable* nicht beim Linken entfernt wurden.

Der ASCII-basierte Browser **links** unter OmniOS tut uns den Gefallen:

```
$ nm /opt/ooce/bin/links | grep get
/opt/ooce/bin/links:

[Index]   Value                 Size                  Type  Bind  Other Shndx   Name
...
[1142]  |              4342472|                     0|FUNC |GLOB |0    |UNDEF  |getaddrinfo
[1157]  |              4344376|                     0|FUNC |GLOB |0    |UNDEF  |getcwd
[1559]  |              4342312|                     0|FUNC |GLOB |0    |UNDEF  |getenv
[2091]  |              4342744|                     0|FUNC |GLOB |0    |UNDEF  |getgrgid
[2067]  |              4344424|                     0|FUNC |GLOB |0    |UNDEF  |gethostname
[1875]  |              4349488|                   499|FUNC |GLOB |0    |14     |getml
[1743]  |              4343640|                     0|FUNC |GLOB |0    |UNDEF  |getpagesize
[1077]  |              4343544|                     0|FUNC |GLOB |0    |UNDEF  |getpid
[841]   |              4663408|                   101|FUNC |LOCL |0    |14     |getpri
[1640]  |              4342760|                     0|FUNC |GLOB |0    |UNDEF  |getpwuid
[1768]  |              4344168|                     0|FUNC |GLOB |0    |UNDEF  |getrlimit
[1094]  |              4341992|                     0|FUNC |GLOB |0    |UNDEF  |getsockname
[2103]  |              4342024|                     0|FUNC |GLOB |0    |UNDEF  |getsockopt
[1684]  |              4343688|                     0|FUNC |GLOB |0    |UNDEF  |gettimeofday
...
```

↪ **links** nimmt hier die modernere Variante: getaddrinfo().

# Schritte bei der 'Name Resolution'

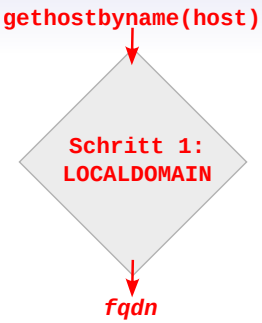
Bei der 'Name Resolution' (über DNS) sind zwei grundsätzliche Akteure zu beachten, die es zu berücksichtigen gilt:

- **Der Unix Kernel**:
  Der besitzt Kenntnisse über die *Netzwerkkonfiguration*, *Routing Tabellen* und vor allem den (nutzbaren) *Interfaces*. Die Interfaces können auch 'virtualisiert' sein, was den Einsatz der von der (DNS) 'Name Resolution' genutzten Systemaufrufe auch in VMs, Jails, oder Docker-Images ermöglicht.
  Der Unix Kernel kann zwar seine Parameter im User-Land Dateisystem ablegen, in dem diese dort 'gespiegelt' werden (`/etc/proc`, **sysctl**), aber benötigt weitere 'Helfer', um Informationen von 'dort' zu beziehen und in seinem Speicher abzubilden und so für alle Nutzbar zu machen.
- **Die *Applikation*, die DNS bzw. 'Name Resolution' Dienste anfordert**:
  Diese muss sich zunächst mit den 'lokalen' Begebenheiten (*Hostname*, *Domainname*) vertraut machen, um den laufenden Kontext kennen zu lernen, und um dann über die DNS-Routinen in den *Libraries* eine 'Name Resolution' zu starten.

Bei der Applikation geschieht dies über folgende Schritte:

1. Ermittlung der lokalen Instanz → **hostname** oder gethostname()
2. Ermittlung der verfügbaren 'Name Resolution' Services → `/etc/nsswitch.conf`
3. Anfordern *lokaler* oder *externe* DNS-Dienste → `/etc/hosts`, `/etc/resolv.conf`

# Schritt 1: Wo lebe ich?




Bei der 'Name Resolution' geht es immer um folgende Fragen:

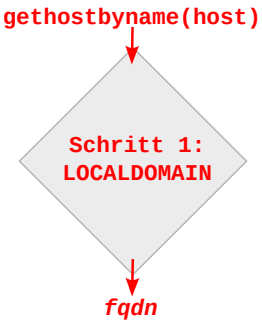
> **Anwendung**: Ich benötige Informationen über '*ho(r)st*' vom Typ [`A, AAAA, PTR, TXT, ...`]! Frage für einen Freund ;-)

'*ho(r)st*' ist hier ein Name ohne Kontext.
Und so könnte die Antwort lauten: *Welchen 'ho(r)st' meinst Du?*

Im DNS-System laut RFC 1034 gibt es das Konzept von *Name Spaces*:

- Unqualifizierte Namen wie '*ho(r)st*' stellen einen lokalen Bezug dar. Dies ist etwa der eigenen '*hostname*'.
- Qualifizierte Namen (*Full Qualified Domain Name*, **FQDN**), stellen eine Kette von (Domain-) Namen bis zur *Root* dar. Namen werden durch Punkte getrennt und stellen ein Label dar. Jedes Label darf maximal 63 Byte (*Oktette*) lang sein und die Gesamtlänge wird auf 255 Byte limitiert.
  Achtung: Der Punkt ist ein *Protokoll-Artefakt* (wie der Punkt bei IPv4-Adressen) und wird bei der eigentlichen Abfrage nicht übermittelt; sondern statt dessen wird ein führendes Byte genutzt, die Länge des folgenden Labels mitzuteilen.
  **Knoten im DNS**: Der Punkt ('node') diente dazu, ein hierarchisches DNS aufzubauen, von dem heute nicht mehr viel übrig geblieben ist.
- Spezialfall liegt vor, falls das letzte Label mit einem Punkt versehen ist: `.com.`
  Dies stellt einen *absoluten Pfad* dar und wird in Notationen wie bei BIND für die Darstellung der *Ressource Records* in der eigenen Zone benutzt.

# Schritt 1: Wie an 'hostname' oder 'domainname' kommen?




Diese Frage ist schwieriger zu beantworten, als es scheint:

- Unsere Anwendung kann aus der C-Lib den Systemaufruf gethostname() benutzen (nicht `gethostbyname()`!).
- Die Anwendung kann nach der (lokal) gesetzten Environment-Variablen `$LOCALDOMAIN` schauen oder aber
- die global gesetzte Datei `/etc/resolv.conf` auswerten.

Die Frage stellt sich: Wie kommt die Information an den richtigen Ort und welcher '`hostname`' ist gemeint?

Im Unix/Linux Umfeld haben wir hierzu eine Reihe von Möglichkeiten:

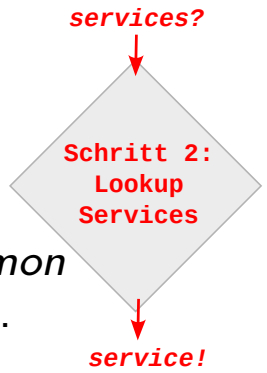
- Das Programm '**hostname**' kann vom Root-User genutzt werden, den *relativen* oder den *FQDN* Hostnamen (neu) zu setzen.
- Der 'Hostname' wird persistiert: `/etc/hostname`, `/etc/init.d/hostname` (Linux).
- Der 'Hostname' muss bei FreeBSD in `/etc/rc.conf` abgelegt sein.
- Der 'Hostname' wird in der Datei `/etc/hosts` eingetragen und (zumindest) mit den IP Loopback-Adressen assoziiert.

↪ Damit ist aber immer noch nicht geklärt, wie wir an das *Domain-Suffix* kommen.

> Das Domain-Suffix hängt davon ab, wo ich meinen Rechner '*einstöpsele*'.

→ *Name Qualification*!

# Schritt 2: Wie an den Name Resolution *Service* herankommen?




Nachdem wir den FQDN für die Name Resolution gebildet haben und somit die *Name Qualification* erfolgt ist, durchläuft unsere Abfrage – typischerweise von gethostbyname() – noch eine *Service Qualification*.

Im Unix-System halt sich noch der Dienst *Name Service Cache Daemon* (**nscd**) als Teil des **nsswitch** Subsystems versteckt – ausser bei Linux. Hierbei wurde die Idee verfolgt, das Lesen von Konfigurationsdateien – die von vielen Diensten benötigt werden – zu beschleunigen, indem sie beim Start des Unix-Systems in den Speicher geladen werden.

→ **nsswitch** ist ein Teil der `clib` (und auch der `glibC`). **man** `nsswitch.conf`:

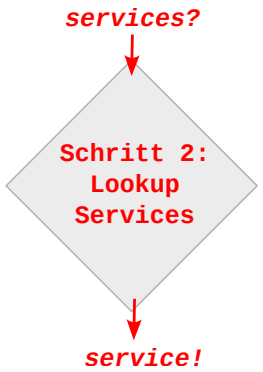
```
NAME
       nsswitch.conf - Name Service Switch configuration file
DESCRIPTION
       The Name Service Switch (NSS) configuration file, /etc/nsswitch.conf, is used
       by  the GNU C Library and certain other applications to determine the sources
       from which to obtain name-service information in a range of  categories,  and
       in  what  order.   Each  category  of information is identified by a database
       name.

       The file is plain ASCII text, with columns separated by spaces or tab charac-
       ters.  The first column specifies the database name.  The  remaining  columns
       describe  the order of sources to query and a limited set of actions that can
       be performed by lookup result.

       The following databases are understood by the GNU C Library:

       aliases     Mail aliases, used by getaliasent(3) and related functions.
       ethers      Ethernet numbers.
       group       Groups of users, used by getgrent(3) and related functions.
       hosts       Host names and numbers, used by gethostbyname(3) and related functions.
```

# Schritt 2: Details der Name Resolution *Services*




Damit läuft das weitere Prozedere wie folgt ab:

- Von gethostbyname() wird zunächst die Datei `/etc/nsswitch.conf` gelesen.
- Der Abschnitt 'host' wird ausgewertet und folgendes gefunden:

```
1 hosts: files dns
```

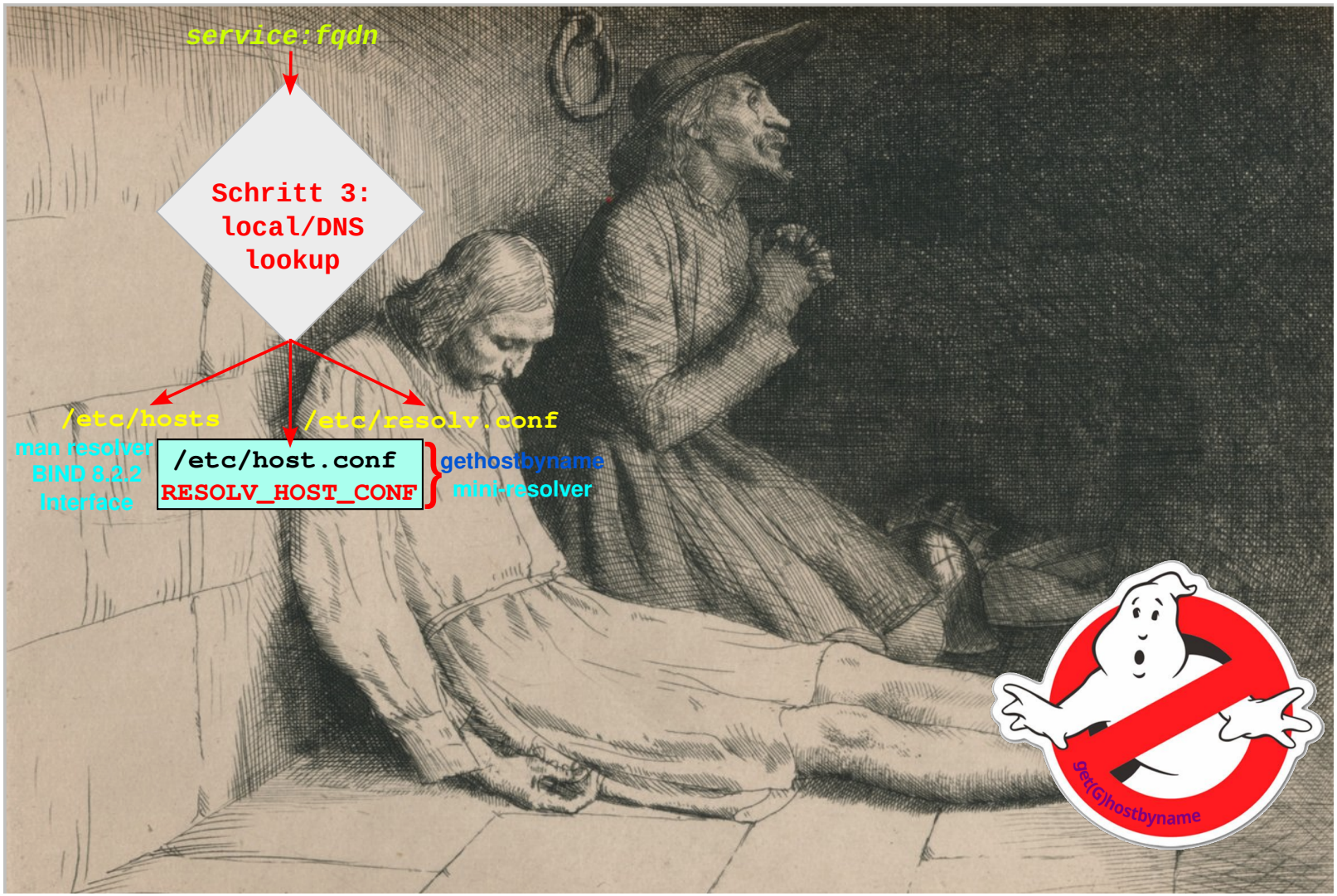
↪ In der `/etc/nsswitch.conf` kann Code abgelegt – also *programmiert* – werden!

Im weiteren Verlauf wird dann die folgende Reihenfolge abgearbeitet:

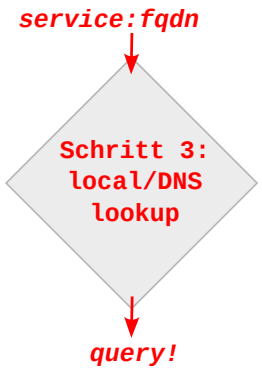
1. Es wird die Datei `/etc/hosts` und hier der von gethostbyname() nachgefragte Name gesucht – und bei einem Treffer, die dort hinterlegte Information als Antwort genutzt.
2. Falls kein Treffer erzielt wurde, wird nun – endlich – das DNS befragt.
   Nur, welches? Hierzu benötigen wir einen *rekursiven Name Server* (Recursor), den wir ansprechen können.

↪ Das ist der Grund, warum Einträge in `/etc/hosts` Vorrang vor DNS-Abfragen besitzen. Wir können hier z.B. unerwünschte FQDN (Domainnamen) 'blackholen', indem wir diesen die IP-Adresse '127.0.0.1' – also *Loopback* – zuweisen.

# Schritt 3: Nutzen der 'eigentlichen' Name Resolution Services



# Schritt 3: Aus dem Name Resolution *Service* Verlies herauskommen




Nun wollen wir uns umschauen und eine Weg finden,
aus dem Verlies befreien, in das wir gestolpert sind.
Bei gethostbyname() gibt es somit drei intrinsische Methoden
einen Aufruf zu bearbeiten:

1. Die Nutzung der Datei `/etc/hosts`; dies ist bei allen Unixen und auch Windows-Systemen gegen und ein Überbleibsel aus der Zeit, wo es das ARPA- bzw. Internet gab; jedoch noch ohne Domain Name Service.
2. Der Einsatz eines 'eigenen' DNS-Resolver über die Datei `/etc/host.conf`, der als BIND Wurmfortsatz angesehen werden kann.
   Bei Linux ist dieser 'deprecated' bei anderen Unix-Systemen komplett entfernt
   → Dessen Arbeitsweise blieb mir im Verlies verborgen.
3. Die Beachtung von `/etc/resolv.conf` und Nutzung der dort hinterlegten Informationen: Der einzig gangbare Pfad aus dem Verlies. Wir werden sehen: Auch der ist steinig und nicht ohne Risiko.

# API change: gethostbyname() ↔ **getaddrinfo**

```
1  struct hostent *gethostbyname(const char *name);
2
3  struct hostent *gethostbyname2(const char *name, int af); // not by OmniOS
4
5  struct hostent {
6    char    *h_name;         /* canonical name of host */
7    char    **h_aliases;     /* alias list */
8    int     h_addrtype;      /* host address type */
9    int     h_length;        /* length of address */
10   char    **h_addr_list;   /* list of addresses */
11 };
```

**Listing:** Beschreibung von gethostbyname() und der zugrunde liegenden struct{}

```
1  int getaddrinfo(const char *hostname, const char *servname,
2                  const struct addrinfo *hints, struct addrinfo **res);
3
4  struct addrinfo {
5    int     ai_flags;       /* AI_PASSIVE, AI_CANONNAME, .. */
6    int     ai_family;      /* AF_xxx */
7    int     ai_socktype;    /* SOCK_xxx */
8    int     ai_protocol;    /* 0 or IPPROTO_xxx for IPv4 and IPv6 */
9    socklen_t ai_addrlen;   /* length of ai_addr */
10   char    *ai_canonname;  /* canonical name for hostname */
11   struct  sockaddr *ai_addr;      /* binary address */
12   struct  addrinfo *ai_next;      /* next structure in linked list */
13 };
```

**Listing:** Beschreibung von getaddrinfo() und der zugrunde liegenden struct{}

↪ Unter FreeBSD gibt es auch das mit vielen Optionen versehene Programm **getaddrinfo** als Pendant zu **host** (und es lügt).

# Das Inventar

Wir haben gesehen, dass der simple (und häufige!) Aufruf von gethostbyname eine ganze Kaskade Dateioperationen nach sich zieht:

1. Es muss der Domainname ermittelt werden und hierfür ggf. die Datei `hostname` konsultiert werden.
2. Als nächstes wird `/etc/nsswitch.conf` gelesen und im Anschluss, sowohl
3. `/etc/hosts` aus auch
4. `/etc/resolv.conf` ausgewertet. Bei jedem Aufruf!

→ Das Problem, dass sich hiermit ergibt, ist, dass diese Dateien nicht nur gelesen, sondern auch geparsed werden müssen. Wie in der Frühzeit von Unix, sind dies *key/value Stores*, die geeignet interpretiert werden müssen.

> Für den Inhalt, den Aufbau und das Auswerten dieser Konfigurationsdateien gibt es keine allgemeingültigen Standards (Syntax, Semantik); weder RFCs noch POSIX bilden diese ab.

↪ Zudem sind die Dateien `/etc/resolv.conf` und zum geringeren Teil `/etc/hosts` *volatil*, wie wir das gleich noch sehen werden.

# Inhalt von /etc/hosts

*Wir können es uns schon denken*:
Es gibt keine einheitliche Syntax für IPv6-Adressen in /etc/hosts:

```
1 # The following lines are desirable for IPv4 capable hosts
2 127.0.0.1       localhost
3 127.0.1.1       thishost.example.org    thishost
4
5 # The following lines are desirable for IPv6 capable hosts
6 ;:1             localhost ip6-localhost ip6-loopback
7 ff02::1         ip6-allnodes
8 ff02::2         ip6-allrouters
```

**Listing:** Debian: (/etc/)hosts man page

```
1 192.9.1.20      gaia                            # John Smith
2
3 # The following is an example of an IPv6 hosts entry:
4 2001:0db8:3c4d:55:a00:20ff:fe8e:f3ad  myhost  # John Smith (not compactified!)
```

**Listing:** OmniOS (/etc/)hosts man page

```
1 # In the presence of the domain name service or NIS, this file may
2 # not be consulted at all; see /etc/nsswitch.conf for the resolution order.
3 #
4 ::1             localhost localhost.my.domain
5 127.0.0.1       localhost localhost.my.domain
```

**Listing:** FreeBSD /etc/hosts

# Inhalt von /etc/resolv.conf

Während `/etc/hosts` eine lokale Datenbank für die Name Resolution (von gethostbyname()) beinhaltet, wird per `/etc/resolv.conf` ein externer Dienst aufgerufen: der *DNS Resolver*.

Um diesen zu konfigurieren und die Arbeit von gethostbyname() zu unterstützen, beinhaltet die Datei `/etc/resolv.conf` einen Satz von Parametern und Optionen:

- `nameserver`: IP-Adressen der externen Resolver/Recurser, bis zu `MAXDNS` (3).
- `domain`: Domain-Name für unqualifizierte Abfragen, wird an *host* angehangen.
- `search`: Wie `domain` aber ggf. mit anderen Domain-Namen.
- `sortlist`: ?? Liste von IPv4 Adressen mit Netzmaske für gethostbyname().
- `options`
  - `debug`: 'Sets `RES_DEBUG` in the `_res.options` field'.
  - `ndots`: Minimale Anzahl der Labels in einem *hostname*.
  - `timeout/retrans`:
  - `attempts/retry`:
  - `rotate`: Rotiere Resolver aus `nameserver` (ansonsten first come/first serve)
  - `reload-period`: Anzahl von Sekunden zum Neuladen von `/etc/resolv.conf` (2 sek)

↪ *Es gibt kein verbindliches Format für* `/etc/resolv.conf`.

> Bei keiner Implementierung ist die Möglichkeit gegeben, den Resolver/Forwarder per IPv6 LLU-Adresse mit dem Interface-Index anzugeben → `fe80::53%eth0`

→ Debian-Linux bietet noch eine weit grössere Anzahl der Parametern, die auch solche für IPv6 und DNSSec umfassen.

# Management von /etc/resolv.conf

Im Gegensatz zur Datei `/etc/hosts`, wird der Inhalt von `/etc/resolv.conf` von einigen Mitspielern verändert; und zwar speziell die Einträge für die `nameserver`:

- In einem Unix/Linux-System sind praktisch alle Netzwerkanwendungen auf den Inhalt von `/etc/resolv.conf` angewiesen, da sie alle den gleichen DNS Sub-Resolver benutzen.
- In einem *managed network* (auch zuhause beim Einsatz einer Fritzbox) werden jedoch die Informationen über die verfügbaren Netzwerkressourcen durch eine zentrale Komponente verteilt, bzw. von dieser bezogen:
  - Dem **DHCPv4-Server** oder
  - dem **IPv6-Router** per *Router Advertisements*, bzw gemeinsam
  - mit dem **DHCPv6-Server**.

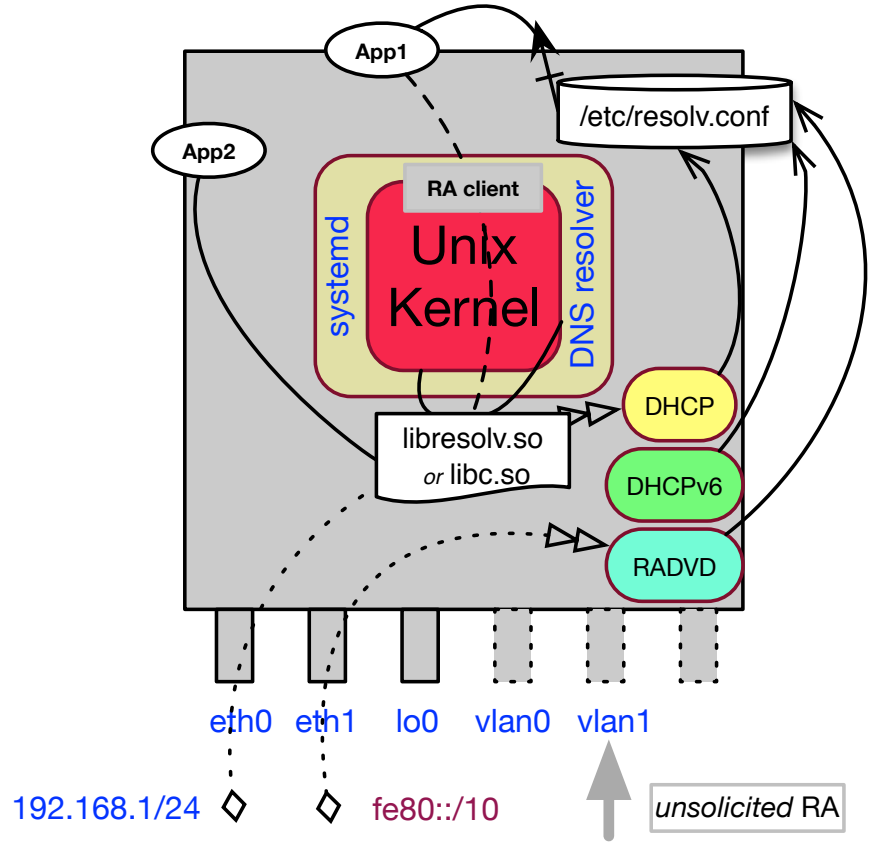



**Abbildung:** Nutzung und Management der Inhalte von `/etc/resolv.conf`

In IPv6 Router Advertisements lasssen sich mittels `RDNSS` die *IPv6-Adressen* der DNS-Resolver sowie per `DNSSL` die *Domain-Kennungen* an die Clients übertragen.

# Network Manager

In einem *managed network* werden für einen Rechner die Netzwerk-Parameter

- *IP-Adresse*,
- *Default Route* (oder bei IPv6 mehrere),
- *DNS Resolver* und
- *Domain Suffix*, sowie
- *Zeit-Server*

wie vorher gezeigt, zentral bezogen und brauchen somit nicht konfiguriert zu werden.

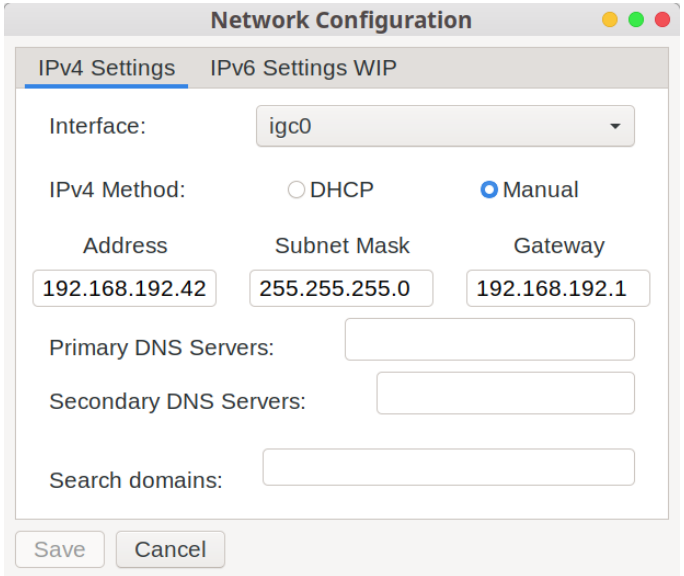



**Abbildung:** Der Network Manager von GhostBSD – ein einfaches Python Skript für den Cinnamon Tray

Komplex wird dieses Setup dann, wenn ein *Roaming User* vorliegt, der mit mehreren Netzen – und hierbei speziell über VPNs – verbunden ist. Um gerade letzteres zu unterstützen, bietet sich der Network Manager an.

↪ Der Network Manager unter FreeBSD kann noch nicht einmal die `/etc/resolv.conf` korrekt auslesen.

> Die dynamische Zuweisung der Parameter in `/etc/resolv.conf` steht in Konkurrenz zur manuellen Konfiguration.

# Openresolv und weitere Lösungen

Die Datei `/etc/resolv.conf` stellt die zentrale Konfigurationsdatei für das Unix-System dar. In einem 'roaming' Szenario ist das häufig nicht die beste Lösung:

1. Per *Interface spezifische Parameter*: Dies ist das Verhalten von Microsoft Windows und kann auch erweitert werden für virtuelle Interfaces.
2. Per *Domain-Suffix spezifische Parameter*: Je nach dem welches Domain-Suffix (z.B. per DHCP mitgeteilt wird, wird der zugehörige Nameserver genutzt.
   → Dies entspricht dem *split-horizon* Verhalten von Nameservern – nun allerdings auf der Client-Seite.
3. *Applikations-spezifische Parameter*: Aktuelle Browser, wie Chrome oder Firefox, nutzen DNS-over-HTTP(S) und setzen daher ihren eigenen Resolver ein.

↪ **openresolv** stellt unter Unix eine Zwischenschicht zum Management der Konfiguration der DNS-Parameter dar.

```
1 # resolv.conf from bge0
2 search foo.com
3 nameserver 1.2.3.4
4
5 # resolv.conf from tap0
6 domain bar.org
7 nameserver 5.6.7.8
```

**Listing:** Beispielkonfiguration von **openresolv** für das zweite Szenario

# Der grosse Bruder: systemd-resolved

Im Linux-Umfeld gibt es einiger Zeit den von *Lennart Poettering* geschaffenen **systemd**-Dienst. Dieser beinhaltetet mittlerweile auch das Modul '**systemd-resolved**':

- Der **systemd-resolved** ist ein eigener Daemon, der auf der IP `127.0.0.53` lauscht,
- Anfragen vom DNS Stub-Resolver entgegen nimmt,
- als *DNS Forwarder* fungiert und die Anfragen weiter leitet,
- zusätzlich eine DNSSec Validierung vornimmt und
- ggf. die DNS Nachrichten per TLS verschlüsselt und zusätzlich
- *Link-Local Multicast Name Resolution* (LLMNR) vornehmen kann.

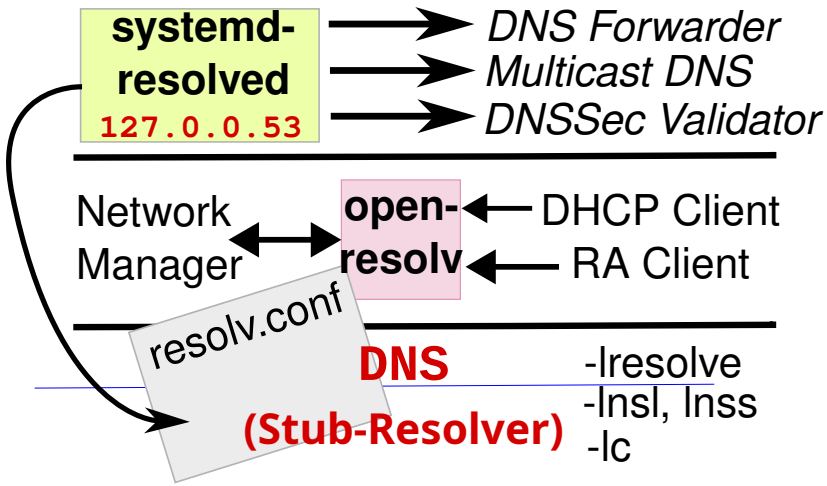



**Abbildung:** Der systemd-resolvd im Zusammenspiel mit dem DNS Stub-Resolver und Openresolv

↪ Der **systemd-resolved** erfüllt im Grunde genommen die gleichen Aufgaben wie die des *Network Service Switch* (NSS) – nun aber als *Daemon* und nicht als *Library*.

# Warum Alternativen?


**Abbildung:** Im DNS Stub-Resolver Morast mit Irrlichtern

# C-Ares DNS Library

**C-Ares** von *Daniel Stenberg* und *Bred House* liefert eine DNS Library auf Grundlage von UDP und TCP und mittels asynchroner Lookups; allerdings ohne DNSSec Validierung:

- c-ares is a modern DNS (stub) resolver library, written in C.
- It provides interfaces for asynchronous queries while trying to abstract the intricacies of the underlying DNS protocol.
- It was originally intended for applications which need to perform DNS queries without blocking, or need to perform multiple DNS queries in parallel.

- One of the goals of c-ares is to be a better DNS resolver than is provided by your system, regardless of which system you use.
- We recommend using the c-ares library in all network applications even if the initial goal of asynchronous resolution is not necessary to your application.

Quelle: `https://c-ares.org/`

# C-Ares DNS Library: Inhalt

- c-ares-1.34.6.tar.gz → 994K
- libcares.a → 2.3M

```
$ size libcares.a
   text    data     bss     dec     hex filename
   1185       0       0    1185     4a1 libcares_la-ares_addrinfo2hostent.o (ex libcares.a)
    861       0       0     861     35d libcares_la-ares_addrinfo_localhost.o (ex libcares.a)
      0       0       0       0       0 libcares_la-ares_android.o (ex libcares.a)
    296       0       0     296     128 libcares_la-ares_cancel.o (ex libcares.a)
    683       0       0     683     2ab libcares_la-ares_close_sockets.o (ex libcares.a)
   2882       0       0    2882     b42 libcares_la-ares_conn.o (ex libcares.a)
   1561       0       0    1561     619 libcares_la-ares_cookie.o (ex libcares.a)
    455       0       0     455     1c7 libcares_la-ares_data.o (ex libcares.a)
   1206       0       0    1206     4b6 libcares_la-ares_destroy.o (ex libcares.a)
    234       0       0     234      ea libcares_la-ares_free_hostent.o (ex libcares.a)
     66       0       0      66      42 libcares_la-ares_free_string.o (ex libcares.a)
    417       0       0     417     1a1 libcares_la-ares_freeaddrinfo.o (ex libcares.a)
   3056       0       0    3056     bf0 libcares_la-ares_getaddrinfo.o (ex libcares.a)
      0       0       0       0       0 libcares_la-ares_getenv.o (ex libcares.a)
   1131       0       0    1131     46b libcares_la-ares_gethostbyaddr.o (ex libcares.a)
   1782       0       0    1782     6f6 libcares_la-ares_gethostbyname.o (ex libcares.a)
   2164       0       0    2164     874 libcares_la-ares_getnameinfo.o (ex libcares.a)
   3985       0       0    3985     f91 libcares_la-ares_hosts_file.o (ex libcares.a)

...
```

**Listing:** Ermittlung der Module der libcares.a

# djbdns Library

Die von Daniel Bernstein für seine Software ucspi-tcp und später für **djbdns** entwickelte **djbdns** Library stelle ich in den **fehQibs** zur Verfügung:

Applikationen, die gegen die `dnscresolv.a` von `djbdnscurve6` der gegen `dnsresolv.a/libdnsresolv.so` von den `fehQlibs` gelinkt sind, erhalten Kenntnis über den DNS Resolver mittels der Environment-Variablen:

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

Hierbei sind mehrere, durch Leerzeichen getrennte `$DNSCACHEIP` und `$LOCALDOMAIN` Angaben möglich.

- IPv6 Adressen werden (immer) im kompatifizierten Format angegen.
- Bei IPv6 LLU Addresses wird der (local) *Interface Name* im kanonischen Format anschliessend an das '%' Zeichen mitgegeben.

↪ Es kann auf die `/etc/resolv.conf` (einschliesslich Interface-Name) zurück gegriffen werden.

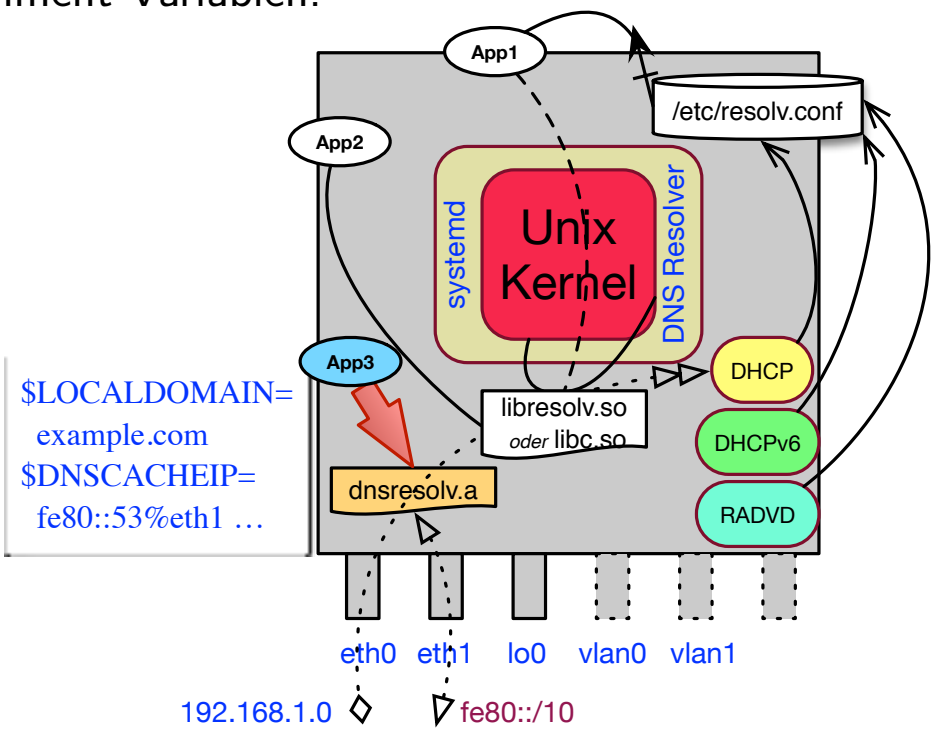



**Abbildung:** Anwendung **App3** nutzt die per Environment provissionierten DNS-Angaben

Meine DNS-Library `dns[c]resolv.a` ist entweder statisch oder dynamisch mit der Applikation zusammen mit den `qlibs.a` gelinkt, die für IPv4/IPv6 Socket-Aufrufe bereit stehen.

# fehQlibs: Inhalt

- fehQlibs-31.tgz → 93K
- libdnsresolv.a → 73K
- libqlibs.a → 168K

```
$ size libdnsresolv.a
   text    data     bss     dec     hex filename
    987       0       0     987     3db dns_domain.o (ex libdnsresolv.a)
    677       0       0     677     2a5 dns_dfd.o (ex libdnsresolv.a)
    375       0       0     375     177 dns_dtda.o (ex libdnsresolv.a)
   2034       0       8    2042     7fa dns_ip.o (ex libdnsresolv.a)
   2887       0      72    2959     b8f dns_ipq.o (ex libdnsresolv.a)
    689       0       8     697     2b9 dns_mx.o (ex libdnsresolv.a)
   1000       0       8    1008     3f0 dns_name.o (ex libdnsresolv.a)
    456       0       0     456     1c8 dns_nd.o (ex libdnsresolv.a)
    706       0       0     706     2c2 dns_packet.o (ex libdnsresolv.a)
   1561       0     224    1785     6f9 dns_random.o (ex libdnsresolv.a)
   1209       0     720    1929     789 dns_rcip.o (ex libdnsresolv.a)
   1617       0      80    1697     6a1 dns_rcrw.o (ex libdnsresolv.a)
    347       0     112     459     1cb dns_resolve.o (ex libdnsresolv.a)
    433       0       0     433     1b1 dns_sortip.o (ex libdnsresolv.a)
   5734       0     128    5862    16e6 dns_transmit.o (ex libdnsresolv.a)
    735       0       8     743     2e7 dns_txt.o (ex libdnsresolv.a)
    645       0       8     653     28d dns_cname.o (ex libdnsresolv.a)
```

**Listing:** Ermittlung der Module der libdnsresolv.a

# Musl C-Library

Der letzte Kandidat, den ich vorstellen möchte, ist die Musl C-library:

- musl-1.2.5.tar.gz → 1.0M

Hier finden sich unter 'network' eine Reihe von DNS-Module zusammen mit den Socket-Funktionen:

```
$ ls
accept.c            gethostbyname.c     htons.c             lookup_serv.c  res_query.c
accept4.c           gethostbyname2_r.c  if_freenameindex.c  lookup.h       res_querydomain.c
bind.c              gethostbyname2.c    if_indextoname.c    netlink.c      res_send.c
connect.c           getifaddrs.c        if_nameindex.c      netlink.h      res_state.c
dn_comp.c           getnameinfo.c       if_nametoindex.c    netname.c      resolvconf.c
dn_expand.c         getpeername.c       in6addr_any.c       ns_parse.c     send.c
dn_skipname.c       getservbyname_r.c   in6addr_loopback.c  ntohl.c        sendmmsg.c
dns_parse.c         getservbyname.c     inet_addr.c         ntohs.c        sendmsg.c
ent.c               getservbyport_r.c   inet_aton.c         proto.c        sendto.c
ether.c             getservbyport.c     inet_legacy.c       recv.c         serv.c
freeaddrinfo.c      getsockname.c       inet_ntoa.c         recvfrom.c     setsockopt.c
gai_strerror.c      getsockopt.c        inet_ntop.c         recvmmsg.c     shutdown.c
getaddrinfo.c       h_errno.c           inet_pton.c         recvmsg.c      sockatmark.c
gethostbyaddr_r.c   herror.c            listen.c            res_init.c     socket.c
gethostbyaddr.c     hstrerror.c         lookup_ipliteral.c  res_mkquery.c  socketpair.c
gethostbyname_r.c   htonl.c             lookup_name.c       res_msend.c
```

↪ Es wäre überaus interessant, diese verschiedenen Ansätze gegeneinander zu benchmarken.

# Zusammenfassung

Wir haben uns einen kleinen Überblick über das Verlies verschafft, in dem sich der DNS Stub-Resolver befindet.

Auch für den Erfahrenen Unix System-Administrator kann der Einsatz des Stub-Resolvers Überraschungen mit sich bringen, da hier viele Mitstreiter unterwegs sind, deren Rollen und Einfluss nicht einfach zu überblicken ist – und vor allem auch gar nicht in RFCs spezifiziert sind.

Mit diesem kurzen Vortrag habe ich versucht – aus dem Verlies kommend – das Dickicht zu lüften und bin dann unversehens in ein Morast mit Irrlichtern vorgestossen.

# Fragen?

# Zusammenfassung

Wir haben uns einen kleinen Überblick über das Verlies verschafft, in dem sich der DNS Stub-Resolver befindet.

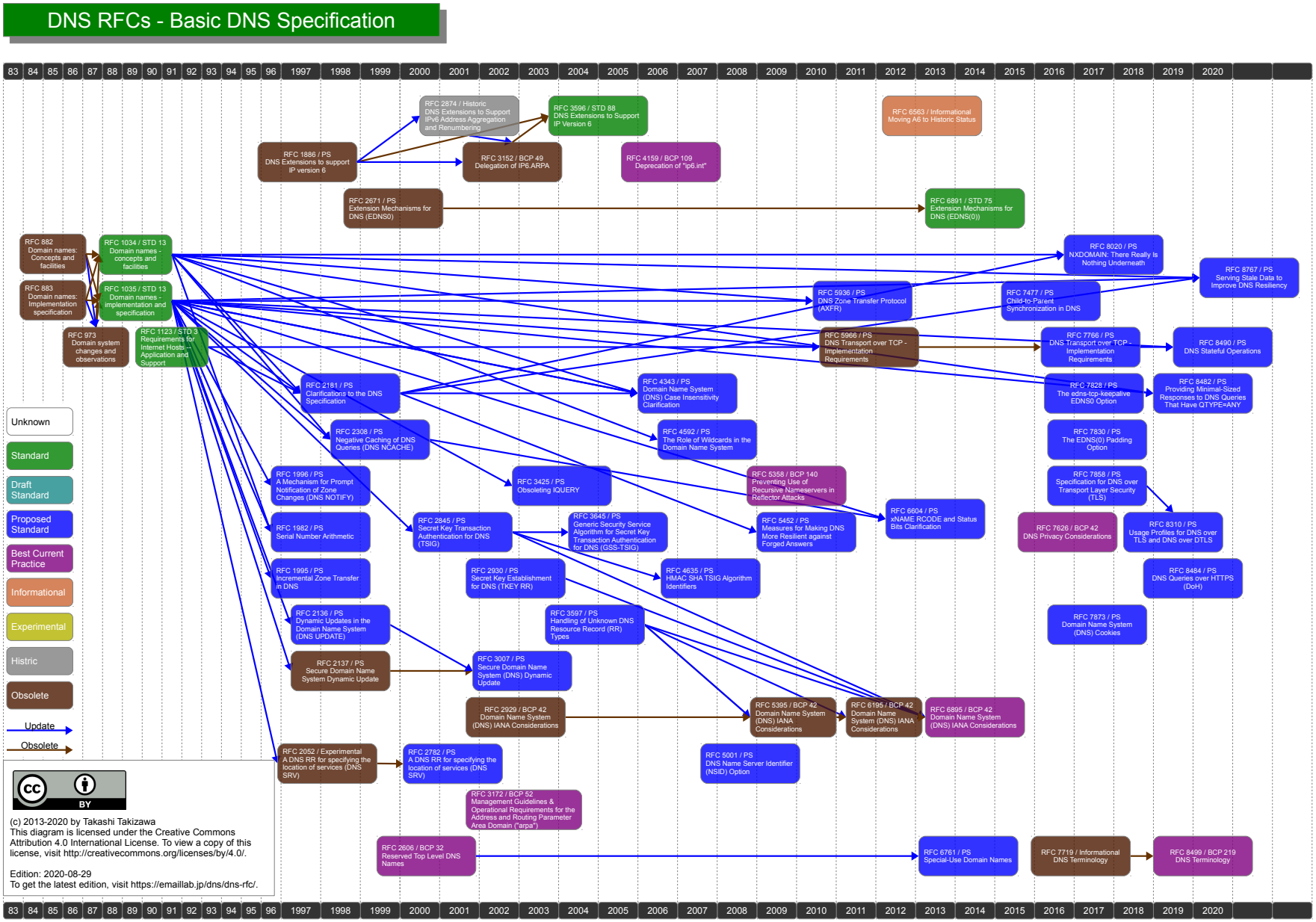
Auch für den Erfahrenen Unix System-Administrator kann der Einsatz des Stub-Resolvers Überraschungen mit sich bringen, da hier viele Mitstreiter unterwegs sind, deren Rollen und Einfluss nicht einfach zu überblicken ist – und vor allem auch gar nicht in RFCs spezifiziert sind.

Mit diesem kurzen Vortrag habe ich versucht – aus dem Verlies kommend – das Dickicht zu lüften und bin dann unversehens in ein Morast mit Irrlichtern vorgestossen.

# Fragen?

# Vielen Dank!

# Zentrale DNS RFCs



# Debian /etc/nsswitch.conf

```
1  #
2  # /etc/nsswitch.conf
3  #
4  # Example configuration of GNU Name Service Switch functionality.
5  # If you have the `glibc-doc-reference' and `info' packages installed, try:
6  # `info libc "Name Service Switch"' for information about this file.
7
8  passwd:         files systemd
9  group:          files systemd
10 shadow:         files
11 gshadow:        files
12
13 hosts:          files dns
14 networks:       files
15
16 protocols:      db files
17 services:       db files
18 ethers:         db files
19 rpc:            db files
20
21 netgroup:       nis
```

**Listing:** Debian: /etc/nsswitch.conf

# OmniOS: /etc/nsswitch.conf

```
1  #
2  # /etc/nsswitch.dns:
3  #
4  # An example file that could be copied over to /etc/nsswitch.conf; it uses
5  # DNS for hosts lookups, otherwise it does not use any other naming service.
6  #
7  # "hosts:" and "services:" in this file are used only if the
8  # /etc/netconfig file has a "-" for nametoaddr_libs of "inet" transports.
9
10 # DNS service expects that an instance of svc:/network/dns/client be
11 # enabled and online.
12
13 passwd:     files
14 group:      files
15
16 # You must also set up the /etc/resolv.conf file for DNS name
17 # server lookup.  See resolv.conf(5). For lookup via mdns
18 # svc:/network/dns/multicast:default must also be enabled. See mdnsd(8)
19 hosts:      files dns mdns
20
21 # Note that IPv4 addresses are searched for in all of the ipnodes databases
22 # before searching the hosts databases.
23 ipnodes:   files dns mdns
24
25 networks:   files
26 protocols:  files
27 rpc:        files
28 ethers:     files
29 netmasks:   files
30 ...
```

**Listing:** OmniOS: /etc/nsswitch.conf

# FreeBSD: /etc/nsswitch.conf

```
1  #
2  # nsswitch.conf(5) - name service switch configuration file
3  #
4  group: compat
5  group_compat: nis
6  hosts: files dns
7  netgroup: compat
8  networks: files
9  passwd: compat
10 passwd_compat: nis
11 shells: files
12 services: compat
13 services_compat: nis
14 protocols: files
15 rpc: files
```

**Listing:** FreeBSD: /etc/nsswitch.conf

# Quellen



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


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


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


[4] *Basic Socket Interface Extensions for IPv6* RFC3493


[5] *IP Version 6 Addressing Architecture* RFC4291


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


[7] *IPv6 Router Advertisement Options for DNS Configuration* RFC6106


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


[9] *DNS Terminology* RFC7719


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


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


[12] *openresolv* https://roy.marples.name/projects/openresolv


[13] *c-ares* https://c-ares.org


[14] *musl libc* https://musl.libc.org/


[15] *Resolving a Hostanme*
https://venam.net/blog/unix/2020/11/01/resolving-a-hostname.html