DNS

Felix von Leitner
CCC Berlin
felix@ccc.de

Dezember 2000

Zusammenfassung

DNS ist nach dem Routing der Schlüssel zum Internet. Dank ICANN hat fast jeder davon gehört, aber verstehen tun es nur wenige. Das will ich ändern.

Einstimmung

„I just upgraded our server to use bind 9.0.1. It is great, fast, and all that. Except for the few Sega Dreamcast users out there that can no longer resolve addresses.“

Von Ash auf **bind-users** gefunden.

Agenda

1. Basis-Eigenschaften des Protokolls
2. Records
3. Das Authority-Konzept
4. CIDR
5. DNSSEC
6. DNS und IPv6

Welches Problem löst DNS?

Problem: User will `ftp.fu-berlin.de`, IP will `130.133.1.100`.

Traditionelle Lösung: `HOSTS.TXT` vom NIC (Network Information Center), vom Admin periodisch per FTP geholt. Enthält Hostnamen („ucbvax“), IPs und Services. Updates per Email.

Probleme

Problem: Keine zwei Rechner durften den gleichen Namen tragen.

Lösung: Hierarchischer Namensraum.

Problem: Zentrale Verwaltung skaliert nicht.

Lösung: Datenbank verteilen.

Problem: Zu viel Traffic.

Lösung: Datenbank verteilen.

DNS: Verteilte Datenbank

Verteilte Datenbank $\neq$ Replizierte Datenbank!

Idee: der Uni-Admin verwaltet die Daten seiner Uni selbständig.

Zentrale Liste verweist auf Uni-Admin.

Folge: immer noch zentral verwaltet!

Zeitlinie

1971: RFC 226 schlägt vor, IPs Namen zuzuordnen.

1981: RFC 799 schlägt das Konzept der *Name Domains* vor (über 400 Nicknames in `HOSTS.TXT`).

1982: RFC 819 führt *fully qualified domain names* ein. Der Dienst *name service* wird vorgeschlagen. Pro Domain soll ein *name server* installiert werden, der Namen zu IP-Adressen umsetzt.

1983: MILNET und ARPANET werden gespalten. RFC 881 schlägt einen Zeitplan für die Umstellung auf Domains vor. Die Rechner im ARPANET erhalten provisorisch `.ARPA` als Domain, die neue Tabelle heißt `DHOSTS.TXT`.

1984: `DHOSTS.TXT` heißt jetzt `HOSTS.TXT`, dieses heißt jetzt `OHOSTS.TXT`.

Zustand vor DNS

Alle Rechner hatten `/etc/hosts`-Dateien, in denen zu IPs der Hostname und die Dienste standen.

Zentrale Verwaltung.

Die Dateien wurden manuell synchronisiert.

Konzept: Domain Name System

Ziel: Einführung eines hierarchischen Namensraumes.

Zweck: Abbildung von Hosts auf IPs und zurück, aber auch von Diensten und Mailboxen.

Unterscheidung: **Name Server** und **Resolver**.

Bessere Unterscheidung: **Name Server**, **Resolver** und **Stub Resolver**.

Konzept: Autorität

Name Server sind für eine Teilmenge des Namensraums zuständig und kennen diese Teilmenge.

Wenn ein Name Server zu seinem Zuständigkeitsbereich befragt wird, antwortet er **autoritativ.**

Konzept: Delegation

Idee: Wenn ein Server nicht zuständig ist, weiß er trotzdem, wer zuständig wäre.

Fall 1: Eintrag ist im Cache

Fall 2: Subdomain

Fall 3: Root Name Server (Default)

„Lame Delegation“

Lame Delegation ist, wenn ein zuständiger Server zu einem Root Server delegiert.

Konzept: Zonen und Records

„Zonen“ sind ein künstliches Layer über Records, das Einsteigern das Verständnis unnötig erschwert.

Idee: `[^. ]*.domain.com` bilden eine Zone.

Umsetzung: Zonen haben einen eigenen `SOA`-Record. Zonen sind die Quanten der Autorität.

djbdns und BIND

Bei den einzelnen Record-Typen kommen Beispiele für BIND und djbdns.

BIND ist für DNS was sendmail für Email ist (d.h. fett, scheiße zu konfigurieren, schnarchlangsam, unzuverlässig und unsicher).

djbdns ist für DNS was qmail für Email ist (d.h. schnell, einfach zu konfigurieren, schnell, zuverlässig und sicher).

Wenn ihr mal einen DNS-Server aufsetzen müßt, nehmt djbdns.

Record-Attribute

Alle DNS-Records haben

1. einen Schlüssel (den Record-Namen, z.B. „`www.domain.com`“)
2. eine Gültigkeit (TTL in Sekunden)
3. eine Klasse (normalerweise „IN“ für das Internet)
4. einen Typen (z.B. „NS“ für Delegation)
5. Daten (kann mehr als ein Feld sein)

Record: SOA

SOA steht für **Start of Authority** und ist ein überflüssiger Legacy-Record, dessen explizite Deklaration leider von BIND gefordert wird.

Atribute: Verschiedene TTLs.

Implizit bei djbdns. Beispiel BIND:

```
$ORIGIN domain.com.
@ 42m40s  IN SOA  ns.domain.com. root.domain.com. (
                  976918270       ; serial
                  4h33m4s         ; refresh
                  34m8s           ; retry
                  1w5d3h16m16s    ; expiry
                  42m40s )                ; minimum
```

Record: NS

Wichtigster Eintrag im DNS ist der `NS`-Eintrag, der auf andere Nameserver verweist. Mit ihm drückt man explizite Delegation aus.

Attribute: Rechnername.

Beispiel BIND:

```
$ORIGIN domain.com.
@           17h36m4s   IN NS   bindsucks
bindsucks   17h36m4s   IN A    10.1.2.3
```

Beispiel djbdns:

```
$ ./add-ns domain.com 10.1.2.3
```

Subdomains

Subdomains werden auch über NS-Records gemacht.

Beispiel BIND:

```
$ORIGIN domain.com.
sub         1D  IN NS  bind.sub
bind.sub    1D  IN A   10.4.5.6
```

Beispiel djbdns:

```
$ ./add-childns sub.domain.com 10.4.5.6
```

Der unsägliche Punkt am Ende

Das Feature, das Admins bei BIND am häufigsten auf den Fuß fällt, ist:

```
bindsucks.domain.com  17h36m4s  IN A   10.1.2.3
```

Wenn kein Punkt am Ende des Schlüssels ist, fügt BIND `$ORIGIN` an, d.h. dieser Record wird als „bindsucks.domain.com.domain.com.“ abgelegt.

Der Punkt am Ende gliedert den Namen explizit der Root-Domain unter, d.h. schaltet die Qualifikation ab. Die meisten Unix-Resolver sind aus BIND-Sourcen extrahiert, daher pingt

```
$ ping www.yahoo.com.
```

www.yahoo.com. Das fällt häufiger CGI-Scripts und Zensur-Proxies auf den Fuß, die Listen von Hostnamen führen, die sie filtern sollen.

Record: A

Bildet Namen auf IPv4-Adressen ab.

Attribute: IP.

Beispiel BIND:

```
www.domain.com.   17h36m4s   IN A    10.1.2.23
```

Beispiel djbdns:

```
$ ./add-host www.domain.com 10.1.2.23
```

Record: PTR

Bildet IPv4-Adressen auf Namen ab („reverse mapping“).

Attribute: Name.

Bei djbdns werden PTR-Records automatisch mit vergeben.

Beispiel BIND:

```
23.2.1.10.IN-ADDR.ARPA.  17h36m4s IN PTR  www.domain.com.
```

(Man beachte den Punkt am Ende von `domain.com` und daß die Reihenfolge der IP umgekehrt wurde!)

Domains mit inkompetenten Hostmastern erkennt man daran, daß PTR-Records fehlen.

IN-ADDR.ARPA?!

Ja. Ist DNS ist nicht toll?

Alles sauber durchdefiniert, kaum Ad-Hoc-Entscheidungen oder gar Legacy-Kram.

Wenn du das schon schlimm findest, warte bis zur IPv6-Sektion!

Reverse SOA Granularität

Ganz schnelle Mitdenker haben sich schon gedacht, daß man auch für den Reverse Lookup einen SOA-Record braucht. Der sieht wie vorher aus, nur daß die Zone jetzt `2.1.10.IN-ADDR.ARPA.` statt `domain.com.` heißt.

Noch schnelleren Mitdenkern ist aufgefallen, daß das ein Problem ist.

Was ist, wenn man nicht für ein ganzes Class C Netz die Autorität hat?

Das geht schlicht nicht. RFC 2317 definiert einen fürchterlich stinkenden Kludge drumherum. Einen Moment Geduld noch, bitte!

Record: CNAME

Bildet Namen auf andere Namen ab (Symlinks für DNS – schlechte Idee!).

Attribute: Name, auf den umgelenkt wird.

Beispiel BIND:

```
kowalewsky  1D  IN CNAME  kowalewski
```

Beispiel djbdns:

```
Ckowalewski.domain.com:kowalewsky.domain.com:86400:
```

Warum CNAMEs scheiße sind

Innerhalb einer Zone sind CNAMEs scheiße, weil sie ineffizienter als A-Records sind.

CNAMEs nach Außen sind scheiße, weil sie ineffizient und unnötige Fehlerquellen sind.

Außerdem stehen dir keine Aussagen über Daten außerhalb deiner Zone zu. Deshalb *haben* wir ja Zonen, damit deine Autorität klar definiert ist.

Record: MX

Delegiert Email-Zuständigkeit (MX = „Mail Exchanger“).

Attribute: Priorität, Name.

Beispiel BIND:

```
domain.com.  17h36m4s  IN MX  10 mail.domain.com.
mail.domain.com.   1D  IN A   10.1.2.42
```

Beispiel djbdns:

```
$ ./add-mx domain.com 10.1.2.23
```

oder

```
@domain.com:10.1.2.23:mail.domain.com.:10:86400:
```

Wildcard Records

Nur der Vollständigkeit halber. **Nicht Benutzen!**

```
*.domain.com.  17h36m4s  IN MX  10 mail.domain.com.
mail.domain.com.   1D  IN A   10.1.2.42
```

Nachteil:

```
$ host cccc.de
cccc.de.domain.com mail is handled (pri=10) by mail.domain.com
```

CIDR Reverse Delegation

Ich verwalte 10.1.2.0-255 und ich möchte meinem Kunden 10.1.2.4-7 vermieten. Sein DNS-Server heißt a.ns.kunde.de und sitzt bei auf der IP 10.1.2.5.

djbdns:

```
$ for i in ‘seq 4 7‘; do
    ./add-childns $i.2.1.10.in-addr.arpa 10.1.2.5
  done
```

und beim Kunden neben den A-Records:

```
$ for i in ‘seq 4 7‘; do
    ./add-ns $i.2.1.10.in-addr.arpa 10.1.2.5
  done
```

CIDR Reverse Delegation mit BIND (RFC 2317)

BIND:

```
$ORIGIN 2.1.10.in-addr.arpa.
4 CNAME 4.4-7
5 CNAME 5.4-7
6 CNAME 6.4-7
7 CNAME 7.4-7
4-7 NS a.ns.kunde.de.
```

und beim Kunden halt die A-Records.

Obskuritäten

Neben diesen Typen gibt es noch TXT, HINFO, MD, MF, MB, MG, MR, NULL, WKS, MINFO, RP, AFSDB, X25, ISDN, RT, NSAP, NSAP_PTR, SIG, KEY, PX, GPOS, LOC, NXT, EID, NIMLOC, SRV, ATMA, NAPTR, TSIG, IXFR, AXFR MAILB, MAILA.

Die CH (Chaos) Klasse stammt aus den 70er Jahren von LAN-Protokoll des MIT. Allerdings:

```
$ dig @10.1.2.3 version.bind. chaos txt
;; ANSWER SECTION:
VERSION.BIND.           0S CHAOS TXT    "4.9.7-REL"
$
```

DNSSEC

RFC 2535 definiert die Record-Typen KEY, SIG und NXT („next“).

KEY speichert öffentliche Schlüssel.

SIG speichert digitale Signaturen.

NXT verweist (zirkulär) auf den nächsten Key. So kann man von einem Angreifer eingefügte Records erkennen.

Als Public Key Verfahren ist DSA Pflicht, RSA/MD5 wird empfohlen.

Das Problem von DNSSEC

DNSSEC ist erst nützlich, wenn die Registries es benutzen und die Domains damit digital signieren.

Das tun sie nicht.

DNSSEC wird von BIND 9 unterstützt, aber es benutzt im Grunde niemand.

Das IPv6 Desaster: AAAA Records

1. Ansatz: AAAA Record.

```
foo  IN AAAA  fe80::260:67ff:fe33:d15b
```

Reverse Lookup über (festhalten)

```
b.5.1.d.3.3.e.f.f.f.7.6.0.6.2.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.\
j.8.e.f.ip6.int IN PTR  foo.domain.com.
```

Problem: Reverse Lookups dauern ewig und drei Tage.

Das IPv6-Desaster: Bitstrings

2. Ansatz: Bitstrings.

Die Hoffnung war, daß damit auch CIDR weniger stark stinkt.

```
$ORIGIN \[x3ffe805002011860/64].ip6.arpa.
\[x0042000000000001/64]  IN PTR  host.domain.com.
```

Ugh. Als wenn man von der normalen Syntax nicht schon Ausschlag bekäme!

Man beachte die tolle neue Domain „ip6.arpa“.

Das IPv6 Desaster: A6 Chains

Szenario: Kunde kriegt IPv6 über ISP1 (Prefix a:b:c::/48, Kundenprefix 17) und ISP2 (Prefix d:e:f::/48, Kundenprefix 23).

```
$ORIGIN kunde.com.
www       A6 64  ::dead:beef:dead:beef cust17.isp1.net.
          A6 64  ::dead:beef:dead:beef net23.isp2.net.
$ORIGIN isp1.net.
cust17    A6 48  0:0:0:17:: ipv6net.isp1.net.
ipv6net   A6 0   a:b:c::
$ORIGIN isp2.net.
net23     A6 48  0:0:0:42:: netipv6.isp2.net.
netipv6   A6 0   d:e:f::
```

Ist doch offensichtlich, oder?! But Wait, There's More!

Das IPv6 Desaster: DNAME

DNAME ist wie CNAME, nur für Domains.

```
$ORIGIN \[x00aa00bbcccc/48].ip6.arpa.
\[xdddd/16]  DNAME  ipv6-rev.example.com.
```

Die Idee ist, daß man alles, das mit dem Prefix `aa:bb:cccc:dddd::/64` beginnt, nach `ipv6-rev.example.com` umleitet.

Das IPv6 Desaster: Reverse Lookup

ISP1:

```
$ORIGIN \[x000a000b000c/48].ip6.arpa.
\[x0017/16]  DNAME  ipv6-rev.kunde.com.
```

ISP2:

```
$ORIGIN \[x000d000e000f/48].ip6.arpa.
\[x0042/16]  DNAME  ipv6-rev.kunde.com.
```

Das IPv6 Desaster: Reverse Lookup

Diese ganzen trivialen Einträge erreichen, daß der Kunde nur eine Zone verwalten muß:

```
$ORIGIN ipv6-rev.kunde.com.
\[xdeadbeefdeadbeef/64]  PTR  www.kunde.com.
```

Wie sehen denn jetzt DNS-Pakete aus?

UDP-Pakete „dürfen“ höchstens 512 Bytes groß sein, danach soll TCP für den Transport benutzt werden.

IPs werden binär gespeichert.

Namen werden „komprimiert“: aus `www.domain.com` wird `3,'www', 6,'domain', 3,'com'` und die einzelnen Komponenten daraus können danach im gleichen Paket referenziert werden.

Es gibt Flags für „Anfrage/Antwort“, „autoritativ“, „rekursiv“.

Eine typische Anfrage

Ich fragte nach www.fefe.de.

```
ich.51960 > sunic.sunet.se.53:  10040 A? www.fefe.de. (29)
sunic.sunet.se.53 > ich.51960:  10040 0/2/2 (108)
ich.63564 > ns.ash.de.53:  18779 A? www.fefe.de. (29)
ns.ash.de.53 > ich.63564:  18779* 1/2/2 A 212.84.209.194 (124)
```

sunic.sunet.se ist für `.de` zuständig. Diese Information hatte mein Resolver schon im Cache.

Das `0/2/2` steht für 0 Antworten (er wußte es nicht), 2 Authority Records (NS-Records, wohin er mich schickt), 2 Glue Records (die IPs zu den NS-Records). Das '*' heißt, daß ns.ash.de autoritativ geantwortet hat. '+' bedeutet, daß die Frage rekursiv war.

Primary und Secondary

Das DNS Protokoll unterscheidet nicht zwischen Primary und Secondary.

Die meisten Registries verlangen zwei DNS-Server (weil BIND so unzuverlässig ist). Das bringt natürlich gar nichts.

Insbesondere sind Antworten des Primary nicht „wertvoller“ als Antworten des Secondary. Es ist nicht einmal definiert, welcher Nameserver zuerst oder überhaupt befragt werden soll.

DNS Probleme

Hauptproblem: DNS ist stateless und benutzt UDP.

D.h. man kann DNS-Pakete spoofen.

Einziger Schutz gegen Spoofing ist ein zufälliges 16-bit Cookie in der Anfrage (oder DNSSEC).

djbdns fragt von einem zufälligen Port ab 1024, d.h. hier kommt noch mehr Zufall ins Spiel. BIND fragt immer von Port 53.

Spoofing ist besonders einfach, wenn man einen Nameserver betreibt, weil man dann über Glue Records Records im Cache der Gegenseite unterbringen kann.

Warum BIND Scheiße ist

BIND hatte zahlreiche Buffer Overflows.

Die BIND-Quellen kommen ursprünglich von Berkeley-Studenten, die das unter einem Grant von Digital gemacht haben. Die haben sich von dem Code später angewidert distanziert, und Paul Vixie hat übernommen.

Auch Paul hat sich zwischenzeitlich von dem Code distanziert.

Mit BIND 9 („complete rewrite“) sollte alles besser werden, aber auch da gab es innerhalb von Wochen das erste Sicherheitsproblem.

Cache Poisoning ist immer noch ein Problem bei BIND.

Warum BIND Scheiße ist

BIND 4-8 schmeißt Records erst aus dem Cache, wenn die TTL abgelaufen ist. Records mit sehr langer TTL sind also ein prima DoS gegen BINDs auf aller Welt.

BIND ignoriert Record-Typen, die er nicht kennt. DNSSEC funktioniert also erst, wenn alle ISPs ihre BINDs upgraden (yeah, right).

BIND antwortet nicht, während Zonen-Dateien neugeladen werden.

BIND macht Müll, wenn eine Zone fehlerhaft ist beim Laden.

```
$ ls -s bind-9.0.1.tar.bz2 djbdns-1.02.tar.gz
2300 bind-9.0.1.tar.bz2
80 djbdns-1.02.tar.gz
$
```

Warum djbdns Scheiße ist

Ich habe noch nichts gefunden.

Ein Korrekturleser kommentierte aber:

*„**Jeder** kann einen DNS-Server schreiben, der besser als BIND ist“*

Zusammenfassung: DNS stinkt

Wer jetzt noch keine Magenverstimmung hat, den lade ich herzlich ein, mir zu helfen, djbdns Support für DNAME, A6 und Bitstrings zu geben!

Diese (und andere) Folien gibt es auf `http://www.fefe.de/`.