Rendezvous mit Linux – zeroconf auf Freien Betriebssystemen

Andreas Krennmair
ak@synflood.at

14. Mai 2004

Zusammenfassung

Dieser Vortrag erklärt die Grundlagen von zeroconf, wie man zeroconf einsetzen kann, und wie eigene Programme zeroconf-fähig gemacht werden können.

Agenda

- Einführung & Motivation
- Theorie link-Lokale IP-Adress-Allokation
- Theorie Multicast DNS
- Theorie Service Discovery
- Was funktioniert unter GNU/Linux?
- Wie kann ich selbermachen?

Einführung & Motivation

- Idee: konfigurationsloser Betrieb von lokalen IP-Netzen, Ad-Hoc Networking
- Teilprobleme:
  - automatische Allokation von IP-Adressen ohne DHCP-Server
  - Übersetzung Namen $\Leftrightarrow$ IP-Adressen ohne DNS-Server
  - automatisches Auffinden von Services im Netz ohne Directory-Server o.ä.
  - bestehende Infrastruktur darf nicht beeinträchtigt werden
- proprietäre Lösungen z.B. in Form von NETBIOS oder AppleTalk zu finden
- $\Rightarrow$ offener Standard musste her

Problemlösungen

Offene Standards zur Lösung dieser Probleme:

- “Dynamic Configuration of Link-Local IPv4 Addresses”
  `http://files.zeroconf.org/draft-ietf-zeroconf-ipv4-linklocal.txt`
- “Multicast DNS”
  `http://files.multicastdns.org/draft-cheshire-dnsext-multicastdns.txt`
- “DNS-Based Service Discovery”
  `http://files.dns-sd.org/draft-cheshire-dnsext-dns-sd.txt`
- definiert von Apple, Microsoft und Sun Microsystems

Theorie zu link-lokaler IP-Adress-Allokation

- aufbauend auf ARP
- von IANA der mittlerweile bekannte `169.254.0.0/16` Adressbereich zugewiesen, die ersten und die letzten 256 Adressen sind allerdings tabu
- Link-lokale IP-Adressen dürfen nicht geroutet werden
- jeder Rechner soll sich selbstständig eine IP-Adresse pro Netzwerk-Interface zuweisen können
- üblicherweise Generierung durch *RNG*
- rechnerspezifische Information (z.B. MAC-Adresse) **sollte** für das Seeding verwendet werden, dass möglichst jedes Mal dieselbe IP-Adressen generiert wird, und damit nicht jeder Rechner dieselbe IP-Adresse generiert

Ablauf der Allokation

- Auswahl einer zufälligen IP-Adresse aus dem `169.254.0.0/16` Adressbereich
- Überprüfung, ob IP-Adresse schon verwendet wird:
  - nach Einschalten, Reboot, Aufwachen aus dem Sleep-Mode, Einstecken des Netzwerkkabels, Aktivieren von WLAN, ...
  - periodische Überprüfung ist verboten, da Verschwendung von Netzressourcen
  - passive Erkennung von Konflikten inkl. passender Reaktion ist vorgesehen
- Überprüfung erfolgt mit Hilfe von sog. “ARP Probe”

ARP Probe

```
struct arp {
  unsigned short hardware_type, protocol_type;
  unsigned char hardware_size, protocol_size;
  unsigned short opcode;
  unsigned char sender_mac[6], sender_ip[4];
  unsigned char target_mac[6], target_ip[4];
};
```

- Absender-IP auf 0.0.0.0 gesetzt
- Empfänger-IP auf zu überprüfende Adresse gesetzt

Konfliktüberprüfung

- Aussenden von 3 ARP-Probes im Abstand von jeweils 1 bis 2 Sekunden
- Empfängt Rechner in der Zeit zwischen Testanfang und 2 Sekunden nach Aussenden der letzten ARP-Probe ein passendes ARP-Paket (Antwort von zu überprüfender Adresse) $\Rightarrow$ Konflikt gefunden
- Bei einem Konflikt muss eine neue Adresse generiert werden
- Nach 10 Konflikten hintereinander darf der Rechner maximal eine Überprüfung pro Minute durchführen, um ARP-Stürme zu verhindern
- Nachdem eine freie IP-Adresse gefunden wurde, muss ARP-Announcement ausgeschickt werden: Sender- und Empfänger-IP sind dabei die ausgewählte IP-Adresse

Konfliktüberprüfung nach Adressauswahl

- Konfliktüberprüfung muss auch nach Auswahl der IP-Adresse weitergeführt werden, es könnte ja ein anderer Rechner versuchen, dieselbe IP auszuwählen
- Bei Konflikt: Verteidigung der eigenen IP, besonders empfehlenswert bei offenen TCP-Connections
- Verteidigung durch Aussenden von ARP-Announcements
- Bei Erkennung von wiederholten Konflikten $\Rightarrow$ Auswahl einer neuen IP, um gegenseitige Verteidung zu vermeiden (ARP-Stürme!)

Link-lokale Adressen bei IPv6

- Link-lokale IPv6-Adressen befinden sich im Bereich `fe80::/64`
- Link-lokale IPv6-Adresse wird aus `fe80`-Prefix und MAC-Adresse zusammengebaut $\Rightarrow$ keine Konfliktüberprüfung nötig, da offiziell jede MAC-Adresse eindeutig ist
- Beispiel:
  - MAC-Adresse: `00:0a:95:66:57:c6`
  - IPv6-Adresse: `fe80::20a:95ff:fe66:57c6`
- Link-lokale IPv6-Adressen dürfen nicht geroutet werden

Multicast DNS (mDNS)

- Beschreibung, wie DNS ohne zentralen DNS-Server in einem LAN geregelt werden kann
- Versenden und Empfangen von DNS-Requests und Replies über Multicast
- Spezielle Multicast-Adresse `224.0.0.251`, Port `5353`
- Top-Level-Domain `.local`:
  - als link-lokal definiert
  - Auswahl von Hosts oder Domains in dieser Domain steht jedem Rechner frei
  - Konflikte als sehr unwahrscheinlich angenommen
  - Konfliktlösung wird bewusst vermieden, da “Konflikte” für Load-Balancing- oder High-Availability-Lösungen verwendet werden könnten

Verhaltenregeln für mDNS

- gecachete Answer Records können an Queries angehängt werden, wenn TTL weniger als bis zur Hälfte abgelaufen ist $\Rightarrow$ könnten für andere Clients nützlich sein
- bemerkt ein Client, dass ein anderer Client denselben Request abgesetzt hat, so setzt er selber keinen mehr ab, sondern verwendet die Antwort, die auch der andere Client bekommt
- Verhaltensregeln vermindern Netzwerkverkehr, ermöglichen es aber auch Clients, die nichts von Multicast wissen, DNS-Requests abzusetzen und interoperabel zu bleiben
- **Dezentrale Architektur stellt Angriffsvektor für u.a. DNS-Cache-Poisoning dar!**

DNS-Based Service Discovery (DNS-SD)

- Problem: es wird ein Service im Netz gesucht, z.B. ein Mailrelay oder die Mp3-Sammlungen von anderen Usern
- Lösung: wir suchen uns den Service in einem Directory
- Verbesserung: wir bauen auf einem bestehenden Directory-Service auf
- Welcher Directory-Service ist jetzt schon allgegenwaertig? **DNS**
  - erprobte Technologie
  - stabile Implementierungen verfügbar
  - quasi überall, wo IP-Connectivity ist, ist auch DNS
- $\Rightarrow$ wir erweitern DNS um die Möglichkeit, zusätzlich auch Services im Netz abfragbar zu machen

Anforderungen an DNS-SD

- nach Services eines bestimmten Typs in einer bestimmten Domain zu suchen
- aufgrund dieser Information IP-Adresse und Port herauszufinden
- Services sollten eindeutig auffindbar bleiben, auch wenn sich IP-Adresse oder Port ändert

DNS-SD im Detail

- DNS-SD baut auf DNS SRV Records (RFC 2782) auf
- Service wird durch speziellen “Hostnamen” identifiziert, z.B. `_http._tcp.example.com`
- Um z.B. alle Webserver im lokalen Netz zu finden, muss man nur einen Abfrage nach `_http._tcp.example.com` stellen, und kriegt prompt eine Liste aller Webserver, die im Service-Directory eingetragen sind
- Jeder Service wird durch `<Instanzname>.<Service>.<Domain>` identifiziert
- Instanzname ist UTF-8-codierte, für Menschen lesbare Bezeichnung des Services (z.B. “Drucker 4. Stock” oder “Andreas’ Urlaubs-Photos”)

Auffinden von Serviceeinträgen im Detail

- exakter Instanzname, Servicetyp und Domain bekannt: Abfrage nach SRV-Record
- dazugehöriger TXT-Record enthält zusätzliche Information in Form von 0 oder mehreren Key-Value-Paaren, z.B.
  `tray=tray4.papersize=a4.orientation=landscape`
- nur Servicetyp und Domain bekannt: Abfrage nach PTR-Record, liefert Liste von SRV-Records zurück

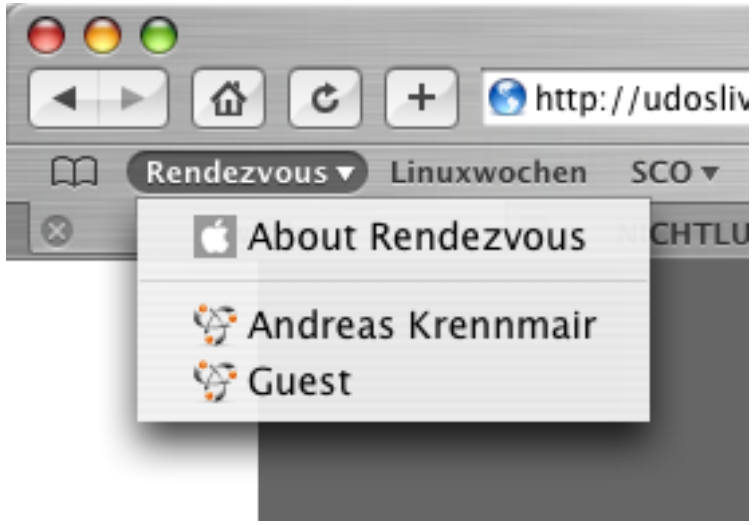
Kombination von link-lokaler Adressierung, mDNS und DNS-SD

- mit link-lokalen IP-Adressen wird Adresse gesucht
- Services werden mit DNS-SD gesucht, Domain ist, da link-lokal, `.local`
- DNS-SD-Abfragen werden mit Hilfe von mDNS versendet
- Damit sind die 3 grundsätzlich voneinander unabhängigen Technologien auf einfache und effiziente Weise miteinander kombiniert, und wir
  - kriegen “automagisch” eine IP-Adresse
  - können Hostnamen in IP-Adressen und umgekehrt ohne zentralen DNS-Server übersetzen
  - können Services im Netz finden
  - beeinträchtigen bestehende Infrastruktur nicht

Zeroconf-Implementierungen

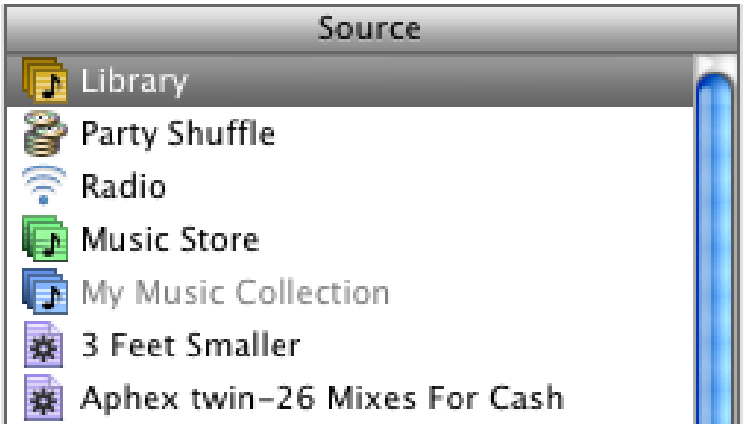
- Erste Implementierung: “Rendezvous”, seit Mac OS X 10.2 integriert und in die meisten Applikationen von Apple eingebaut
- Diese Implementierung ist Open Source (APSL-Lizenz), und von Apple bewusst portabel gehalten: compiliert und läuft auf Linux, FreeBSD, NetBSD, OpenBSD, Solaris, VxWorks, Mac OS 9, Windows
- Problem: APSL inkompatibel zur GPL
- Alternative: Howl
- Howl ist GPLed und läuft unter Linux, FreeBSD, Mac OS X und Windows
- Weitere Alternative: Linux Zeroconf Project: noch nicht standardkonform, unvollständig

Beispiele für gute Integration von Service Discovery (1)



Safari, der auf Konqueror basierende Browser von Apple, zeigt alle Webseiten im lokalen Netz an. Apache mit dem Module `mod_rendezvous` (standardmäßig bei OSX dabei) announced automatisch beim Start alle Seiten, die von Usern in `$HOME/Sites` abgelegt wurden.

Beispiele für gute Integration von Service Discovery (2)



iTunes, der Audioplayer von Apple, entdeckt automatisch im Netz freigegebene Musiksammlungen, sodass mehrere User übers lokale Netz gegenseitig ihre Musik hören, jedoch nicht kopieren, können (zumindest nicht mit iTunes selbst *fg*).

Wie gut ist zeroconf/Rendezvous in Linux integriert?

- Leider quasi noch überhaupt nicht
- Bis jetzt hat nur SuSE den ausgelieferten Resolver mDNS-fähig gemacht, wobei mDNS bei der `.local`-Domain eingesetzt wird. In den ChangeLogs wird das als “non-standard extension” bezeichnet. :-/
- Mandrake setzt AFAIK auf die IETF-Konkurrenztechnologie SLP

Wie zeroconf/Rendezvous selbermachen unter Linux?

- Howl einsetzen: liefert einige Utilities, mit denen Services einfach announced werden können
- Applikationen selbst anpassen

Beispiel 1: mit Howl einen Webservice announcen

- Schritt 1: `mDNSResponder` starten. `mDNSResponder` kümmert sich um das Announcen von Services sowie das Stellen und Beantworten von Anfragen
- Schritt 2: Service mit `mDNSPublish` announcen

```
root# mDNSResponder          # startet den mDNSResponder-Daemon
user$ mDNSPublish "Subversion Repository" _http._tcp 80 path=/svn/
Subversion Repository _http._tcp 80
publish reply: Started
user$ mDNSBrowse _http._tcp
browse reply: Add Service Subversion Repository _http._tcp. local.
resolve reply: Subversion Repository _http._tcp. local. 192.168.23.6 80
path=/svn/
key = path, data is 5 bytes
```

Beispiel 2: mit Howl `mod_rendezvous` simulieren

```
#!/bin/bash
mDNSPublish `hostname` _http._tcp 80 &
for HOMES in /home/*/public_html ; do
  USERS=`echo $HOMES | cut -d "/" -f 3`;
  mDNSPublish "$USERS's website" _http._tcp 80 "path=~$USERS" &
done
```

(c) 2004 Michael Bauer <mihi@ch3cooh.org>

Programmieren mit Howl

- Howl bietet eine umfangreiche API, um Service Announcement und Discovery zu tätigen
- Leider sehr komplex, API noch nicht stabil
- Howl bringt einige leicht verständliche Beispiele mit
- Einführung würde Rahmen des Vortrages sprengen

Danke für die Aufmerksamkeit!

Noch Fragen?
Wichtige Adressen:

```
http://www.zeroconf.org/
http://www.multicastdns.org/
http://www.dns-sd.org/
http://www.dns-sd.org/ServiceTypes.html
http://www.porchdogsoft.com/products/howl/
http://zeroconf.sourceforge.net/
http://developer.apple.com/darwin/projects/rendezvous/
```

...to shape teh future!