

Humboldt-Universität zu Berlin



Institut für Informatik

HK Spezielle Techniken der Rechnerkommunikation

Zero Configuration Networking

Thomas Kollbach, Christian Beier

16. Juli 2007

Diese Arbeit soll einen Überblick über die verschiedenen Technologien der Autokonfiguration der Vermittlungs- wie auch Sicherungsschicht geben. Es werden Technologien vorgestellt, die auf verschiedenen Schichten des OSI-Modells arbeiten und dazu beitragen, den Einsatz von Netzwerken ohne oder mit geringer vorheriger Konfiguration zu ermöglichen.

Es werden sowohl IPv4 als auch IPv6 betrachtet.

Inhaltsverzeichnis

1	Einleitung	2
1.1	Anwendungsgebiete	2
1.2	Komponenten	3
2	Techniken im Überblick	3
2.1	Alte Welt - IPv4	3
2.1.1	Adresskonfiguration	4
2.1.2	Namensauflösung	15
2.1.3	Diensterkennung	20
2.2	Neue Welt - IPv6	26
2.2.1	Adresskonfiguration	30
2.2.2	Namensauflösung	36
3	Experiment	39
4	Fazit	42
	Literatur	44

1 Einleitung

Das Verwenden und insbesondere das Konfigurieren von Netzwerkhardware und -software ist eine komplexe und mitunter langwierige Aufgabe auch für Experten. Mit der zunehmenden Verbreitung von Netzwerktechnologie in Privathaushalten, der gewachsenen Mobilität von netzwerkfähigen Geräten sowie der Vernetzung von klassischen „Consumer Electronic“-Geräten, wie etwa Spielkonsolen, Set-Top-Boxen oder DVR-Geräten, wachsen die Anforderungen an Netzwerktechnologie, konfigurationsarm oder gar konfigurationslos zu funktionieren.

Das momentane Hauptanwendungsgebiet von Technologien zur Autokonfiguration sind kleine und mittlere Netzwerke sowie isolierte Segmente in größeren Netzwerken. Bestimmte Autokonfigurationstechniken, etwa das Dynamic Host Configuration Protocol (DHCP), werden heute beinahe flächendeckend in Netzwerken eingesetzt, so daß die Netzwerkwelt mittlerweile ohne sie gar nicht denkbar wäre. Andere, etwa die in IPv6 enthaltenen Autokonfigurationstechnologien, sind konzeptuell und technisch fortgeschrittener, werden aber kaum eingesetzt. Wieder andere, etwa das Peer Name Resolution Protocol (PNRP) von Microsoft oder Apples mDNS/DNS-SD-Stack, sind herstellerspezifische Insellösungen, die trotz offener Standards von anderen Herstellern noch kaum adaptiert wurden.

1.1 Anwendungsgebiete

So vielfältig wie die Technologien zur Autokonfiguration sind auch die Anwendungsgebiete:

- Ad-Hoc Netze:

Netzwerke mit häufig wechselnden Teilnehmern, etwa WLAN-Hotspots an öffentlichen Plätzen. Diese Netzwerke sind ohne Autokonfigurationstechniken wie DHCP nicht denkbar.

- Private Heimnetzwerke:

Die zunehmende Vernetzung von Privatwohnungen konfrontiert Personen ohne fundiertes Wissen über Netzwerke mit Netzwerktechnologie. Um die Hürde zum Einsatz von Netzwerkfunktionen in Consumer-Electronics-Geräten niedrig zu halten, werden in diesen zunehmend Autokonfigurations-Protokollstacks verwendet. Ein Beispiel hierfür ist die mDNS/DNS-SD Unterstützung von Musikdateiservern.

- Vereinfachung der Administration großer Netze:

Autokonfigurationstechnologien wie DHCP werden von Administratoren großer Netze oft eingesetzt, um den Administrationsaufwand zu verringern. Auch IPv6 wird hier zukünftig eine große Rolle spielen.

1.2 Komponenten

Autokonfigurationstechniken sind zahlreich und finden Anwendung in den verschiedensten Gebieten. Ein vollständiges Autokonfigurationssystem besteht aus einem Stapel von Komponenten, die verschiedene Teilaufgaben erfüllen und damit das Gesamtsystem bilden:

- Addressverwaltung:

Damit Netzwerkkomponenten kommunizieren können, benötigen diese zunächst eine eindeutige Adresse. Die bei weitem größte Zahl der Autokonfigurationssysteme enthält eine Komponente zur eindeutigen Vergabe von Netzwerkadressen.

- Namensauflösung:

Zur Vereinfachung des Einsatzes von Netzwerkkomponenten ist es üblich, einen Dienst zur Abbildung von Namen zu Netzwerkadressen zu verwenden.

- Dienst-Ankündigung und -Erkennung:

Eine zentrale Komponente vieler Autokonfigurationssysteme ist die Dienstankündigung. Diese ermöglicht es, je nach System, Dienste im Netzwerk bekannt zu machen, zu suchen und auch zu nutzen.

2 Techniken im Überblick

Dieser Abschnitt soll einen Überblick über die verschiedenen Autokonfigurationstechniken geben. Er ist in IPv4 („Alte Welt“) und IPv6 („Neue Welt“) gegliedert, da beim Design von IPv6 Konzepte der Autokonfiguration von Anfang an mit eingeflossen sind.

Verzichtet wird dabei auf eine nähere Besprechung von UPnP, das auch unter Autokonfigurationstechnologien zu fassen ist.

2.1 Alte Welt - IPv4

Die Version 4 des Internet Protocol enthält keine Technologien oder Ansätze zur Autokonfiguration, was hauptsächlich auf das Alter der Spezifikation zurückzuführen ist. Als IPv4 entwickelt wurde, waren Netzwerke noch recht selten und klein. Es war schwer vorstellbar, dass IP-Netze einmal Millionen von Computern umfassen und in vielen (Privat-)Häusern präsent sein könnten. Netzwerktechnologie war zur Zeit der Konzeption von IPv4 noch eine Spezialtechnologie, die von Experten gewartet wurde.

Aus diesem Grund wurden *nachträglich* Protokolle und Stacks eingeführt, die eine Auto-konfiguration von Netzwerken ermöglichen. Alle diese Stacks benötigen eine Komponente zur automatischen Adresskonfiguration und -verteilung.

Mit dem Wachstum der IPv4-Netze (und insbesondere des Internet), werden mehr Tech-nologien zur Autokonfiguration möglich und auch nötig.

2.1.1 Adresskonfiguration

RARP

Das „Reverse Address Resolution Protocol“ (RARP) ist ein Protokoll, dass die Zuord-nung von Adressen des IP(v4)-Protokolls auf Basis der Ethernet-MAC-Adressen der Netzwerkteilnehmer ermöglicht.

Es stammt aus dem Jahre 1984.

Funktionsweise RARP arbeitet auf Layer-3 des OSI-Schichtmodells (Internetwork-Layer) und verwendet einen Ethernet-Broadcast, um ein Ethernet-Paket an alle Teilneh-mer des lokalen Ethernets zu schicken, in der Hoffnung, daß ein RARP-Server antwortet.

Paketformat Im Ethernet-Frame schließt sich das RARP-Paket an den MAC-Header und damit an das Ethernet-Typfeld an. Der Ethernettyp für RARP ist 0x8035. Der unterschiedliche Ethernet-Typ unterscheidet sich damit vom Verwandten ARP, so daß ARP-Server nicht von RARP-Paketen und umgekehrt erreicht werden.

0	8	16	24
Hardwareadrestyp (0x1)		Protokolladrestyp (0x800)	
Hardwareadressgröße (0x6)	Protokolladressgröße (0x4)	Operation	
Quell-MAC-Adresse			
Quell-MAC-Adresse		Quell-IP-Adresse	
Quell-IP-Adresse		Ziel-MAC-Adresse	
Ziel-MAC-Adresse			
Ziel-IP-Adresse			

- **Operation** (2 Byte) enthält den Wert, der angibt, welche Operation ausgeführt werden soll (3 für RARP Request, 4 für RARP Reply).
- **Quell-MAC-Adresse** (6 Byte) enthält in einem RARP Request-Paket die MAC-Adresse des Senders. In einem RARP Reply enthält es die MAC-Adresse des ant-wortenden Servers.

- **Quell-IP-Adresse** (4 Byte) ist bei einem RARP Request undefiniert. In einem RARP Reply enthält es die IP-Adresse des antwortenden Servers.
- **Ziel-MAC-Adresse** (6 Byte) enthält in einem RARP Request-Paket die MAC-Adresse des Senders. In einem RARP Reply enthält es die MAC-Adresse des anfragenden Hosts.
- **Ziel-IP-Adresse** (4 Byte) ist bei einem RARP Request undefiniert. In einem RARP Reply enthält es die IP-Adresse des anfragenden Hosts.

Das Hauptanwendungsgebiet von RARP ist in Verbindung mit TFTP, dem Trivial File Transfer Protocol, dass zum starten von Festplattenlosen Thin-Clients verwendet wird.

Beschränkungen Die Verwendung von RARP ist nur in lokal begrenzten Ethernet-Netzen möglich, es ist sozusagen Link-Local. Zusätzlich muss die Zuordnung von MAC- zu IPv4-Adressen auf einem zentralen Server gepflegt werden. Außerdem erlaubt RARP nur das Setzen einer IP-Adresse, so das andere Optionen, wie etwa Subnetzmasken, Gateways oder Optionen für Dienste der Applikationsschicht (etwa DNS-Server) über RARP nicht vermittelt werden können.

Sicherheit Jeder mit physikalischem Zugang zum Netz kann einen RARP-Server aufsetzen, der dann falsche oder doppelte IP-Adressen zurückgibt oder auf einen anderen TFTP-Server verweist (welcher wiederum andere Boot-Images verteilt).

[RARP-RFC, RARP-Wikipedia]

BOOTP

Das Bootstrap Protocol (BOOTP) ist ein Protokoll in der Applikationsschicht (7. Schicht im OSI-Modell), das verwendet wird, um einem Netzwerkteilnehmer eine IPv4-Adresse und zusätzliche Parameter zu übergeben. Es ist, obwohl auf einer anderen Netzwerkschicht angesiedelt, das Defacto-Nachfolgeprotokoll für RARP. Es wurde 1985 im RFC 951 definiert und 1993 in RFC 1497 erweitert.

Funktionsweise Die Kommunikation bei Verwendung von BOOTP besteht lediglich aus einer Anfrage und einer Antwort, beides in Form eines UDP Pakets.

Die Anfrage (Opcode-Feld = 1, *Bootrequest*) wird als netzweiter Broadcast (Zieladresse 255.255.255.255) oder mit der Bootp-Serveradresse - sofern bekannt -, mit der Quelladresse 0.0.0.0 bzw. der eigenen IP-Adresse, sofern diese schon bekannt ist, an Port 67 gesendet.

Die Antwort (Opcode-Feld = 2, *Bootreply*) wird je nachdem, ob der Client seine IP-Adresse schon kannte, mit einem Broad- oder Unicast an Port 68 des Clients zurückgesandt

und enthält

0	8	16	24
OP	HTYPE	HLEN	HOPS
XID			
SECS		unbenutzt	
CIADDR			
YIADDR			
SIADDR			
GIADDR			
CHADDR (16 Byte)			
SNAME (64 Byte)			
FILE (128 Byte)			
VEND (64 Byte)			

- **OP:** packet op code / message type. 1 = BOOTREQUEST, 2 = BOOTREPLY
- **HTYPE:** hardware address type, nach „Assigned Numbers“ RFC.
- **HLEN:** hardware address length, nach „Assigned Numbers“ RFC.
- **HOPS:** (optional) Verwendet beim Cross-Gateway-Booten
- **XID:** Zufallszahl zur Zuordnung von BOOTP-Nachrichten, da ja evtl. keine eindeutige Quell- und Zieladresse verfügbar ist
- **SECS:** Sekunden seit der Client gebootet hat
- **CIADDR:** Client-IP-Adresse, ausgefüllt wenn bekannt
- **YIADDR:** IP-Adresse die dem Client zugewiesen wird, nur verwendet, wenn CIADDR 0
- **SIADDR:** IP-Adresse des BOOTP Servers
- **GIADDR:** (optional) Gateway IP-Adresse
- **CHADDR:** MAC-Adresse des Clients, vom Client ausgefüllt
- **SNAME:** Server Hostname als Null-Terminierter-String
- **FILE:** Pfad der Boot-Datei zum Starten des Clients
- **VEND:** (optional) Herstellerspezifische Angaben

In allen Fällen wird ein Timeout verwendet, um Fehlübertragungen zu erkennen.

Sicherheit BOOTP basiert UDP und IP auf und ist damit schon inhärent anfällig für z. B. Spoofing-Attacken. Noch dazu verteilt BOOTP netzsensitive Konfigurations-Informationen. Dadurch eröffnen sich vielfältige Angriffsmöglichkeiten:

Es ist leicht, unauthorisierte BOOTP-Server aufzusetzen. Diese könnten dann eigene Boot-Images verteilen oder falsche und potenziell kommunikationsunterbrechende Informationen an Clients senden, beispielsweise falsche oder doppelte IP-Adressen, inkorrekte Routing-Informationen oder falsche DNS-Server-Adressen.

Böswillige BOOTP-Clients könnten sich als legitime Clients ausgeben und Informationen einholen, die eigentlich nicht für sie bestimmt sind, inklusive der Boot-Images. Außerdem könnten solche Angreifer alle Ressourcen für sich beanspruchen und dadurch anderen Clients verweigern.

[BOOTP-RFC, BOOTP Extensions-RFC]

DHCP

Das Ziel von DHCP ist die vollautomatische Einbindung eines neuen Computers in ein bestehendes Netzwerk ohne weitere Konfiguration zu ermöglichen. Es verwendet dabei ein RARP ähnliches, aber deutlich erweitertes Protokoll, das es vor allem Ermöglicht weit mehr Konfigurations-Optionen an den Client zu übertragen. Der Einsatz von DHCP ist heute aus Netzwerken nicht mehr wegzudenken und ist eine Grundlage der Kommunikation in vielen Netzen. Es baut auf BOOTP auf und wurde erstmals 1993 in RFC 1531 und 1541 spezifiziert und 1997 in RFC 2131 überarbeitet.

Paketformat DHCP erweitert das BOOTP um verschiedene Funktionen. Daher ist ein DHCP-Paket nur eine erweiterte Version eines BOOTP-Pakets und ist ebenfalls in UDP gekapselt. Auch die Kommunikation in Form von Frage/Antwort erfolgt analog zu BOOTP, wobei das op-Feld verwendet wird um die Quelle der Nachricht (Server bzw. Client) zu signalisieren, während die Protokoll-spezifische Nachrichtentyp im Feld „DHCP Request Type“ im Optionsfeld festgelegt wird.

Das Paketformat von DHCP:

0	8	16	24
OP	HTYPE	HLEN	HOPS
XID			
SECS		FLAGS	
CIADDR			
YIADDR			
SIADDR			
GIADDR			
CHADDR (16 Byte)			
SNAME (64 Byte)			
FILE (128 Byte)			
OPTIONS (312 Byte)			

- **OP:** packet op code / message type. 1 = REQUEST, 2 = REPLY
- **HTYPE:** Netztyp (z. B. 6 = IEEE 802 Netzwerke oder 7 = ARCNET)
- **HLEN:** Länge der physikalischen Netzadresse in Bytes (z. B. 6 = MAC/Ethernet-Adresse)
- **HOPS:** (optional) Anzahl der DHCP-Relay-Agents auf dem Datenpfad
- **XID:** ID der Verbindung zwischen Client und Server
- **SECS:** Sekunden seit der Client gebootet hat
- **FLAGS:** zur Zeit ist nur das erste Bit verwendet (zeigt an, ob der Client noch eine gültige IP-Adresse hat), die restlichen Bits sind für spätere Protokollerweiterungen reserviert
- **CIADDR:** Client-IP-Adresse, wenn bekannt
- **YIADDR:** IP-Adresse die dem Client zugewiesen wird, nur verwendet, wenn CIADDR 0
- **SIADDR:** IP-Adresse des DHCP-Servers
- **GIADDR:** Relay-Agent-IP-Adresse
- **CHADDR:** MAC-Adresse des Clients, vom Client ausgefüllt
- **SNAME:** (optional) Servername als nullterminierter String
- **FILE:** (optional) Pfad einer Datei (z. B. System-Kernel), welche der Client via TFTP herunterladen kann
- **OPTIONS:** (optional) DHCP-Parameter und -Optionen

Die Pakete unterscheiden sich von BOOTP-Paketen hauptsächlich durch das größere Optionsfeld. Die ersten vier Bytes des Optionsfelds eines DHCP-Paketes enthalten die (Dezimal-)Werte 99, 130, 83 und 99, um das Optionsfeld zu signalisieren (Analog zu BOOTP aus RFC 1497). Das Optionsfeld muss die „DHCP Message Type“ Option enthalten, die einen von vier DHCP-Pakettypen definiert. Der Rest der Felder ist, in Abhängigkeit von diesem Feld, entweder notwendig oder optional. Die verschiedenen „DHCP Message Types“ sind

- **DHCPDISCOVER:** Ein Client ohne IP-Adresse sendet eine Broadcast-Anfrage nach Adress-Angeboten an den/die DHCP-Server im lokalen Netz
- **DHCPOFFER:** Der/die DHCP-Server antworten mit entsprechenden Werten auf eine DHCPDISCOVER-Anfrage
- **DHCPREQUEST:** Der Client fordert (eine der angebotenen) IP-Adresse(n), weitere Daten sowie Verlängerung der Lease-Zeit von einem der antwortenden DHCP-Server
- **DHCPACK:** Bestätigung des DHCP-Servers zu einer DHCPREQUEST-Anforderung
- **DHCPNAK:** Ablehnung einer DHCPREQUEST-Anforderung durch den DHCP-Server
- **DHCPDECLINE:** Ablehnung durch den Client, da die IP-Adresse schon verwendet wird
- **DHCPRELEASE:** Der Client gibt die eigene Konfiguration frei, damit die Parameter wieder für andere Clients zur Verfügung stehen
- **DHCPINFORM:** Anfrage eines Clients nach Daten ohne IP-Adresse, z. B. weil der Client eine statische IP-Adresse besitzt

Zusätzlich sind noch viele Optionen insbesondere für DHCPOFFER definiert, mit denen der Server unter anderem folgende Konfigurationsoptionen an den Client weiterreichen kann:

- IP-Adresse und Netzwerkmaske
- Default-Gateway
- DNS-Nameserveradresse
- WINS-Server
- Proxy-Konfiguration via WPAD (Web Proxy Autodiscovery Protocol) - Protokoll zur automatischen Bekanntgabe eines HTTP-Proxies.

Funktionsweise Ein DHCP-Server kann Adressen auf verschiedene Weisen vergeben, manuell oder automatisch zugeordnet:

Manuelle Zuordnung Bei der manuellen Zuordnung werden IP-Adressen (und andere Parameter) bestimmten MAC-Adressen fest und auf unbestimmte Zeit zugeordnet. Bei dieser Variante können keine weiteren Clients dem Netz beitreten, was unter Sicherheitsaspekten allerdings auch ein Vorteil sein kann.

Automatische und dynamische Zuordnung Hier wird am DHCP-Server ein fester Bereich von IP-Adressen definiert, den der Server an Clients vergeben kann. Allerdings haben auch hier die Zuordnungen unbegrenzte Gültigkeit. Das ist erst anders bei der dynamischen Zuordnung, bei der eine IP-Adresse nur für eine gewisse „Lease-Time“ vergeben wird.

Ablauf der Kommunikation Da bei DHCP die Kommunikation wie bei BOOTP hauptsächlich über Broadcasts abläuft, ist es eigentlich - wie BOOTP - auf eine Broadcast-Domäne, sprich ein Ethernet-Linksegment beschränkt. Bei DHCP ist es aber möglich, Anfragen mittels sogenannter DHCP-Relays über Netzwerksegment-Grenzen hinweg weiterzugeben.

Die Kommunikation ähnelt sehr dem Ansatz, den schon BOOTP verwendete.

Will ein Client seine IP-Adresse erfahren, schickt er ein DHCPDISCOVER (mit seiner MAC-Adresse) als Broadcast an die IP-Adresse 255.255.255.255, Port 67. Die Absenderadresse ist die unspezifizierte IP 0.0.0.0, Port 68. Der oder die Server antworten mit DHCPOFFER (wiederum als Broadcast) und machen Vorschläge. Der Client kann nun unter den Vorschlägen einen auswählen und seine Wahl mittels DHCPREQUEST dem Server bekanntgeben. Eventuell vorhandene andere Server werten dies als Absage. Der Server kann nun mit DHCPACK bestätigen oder die Wahl mit DHCPNAK verweigern.

Nach Ablauf der Hälfte der Lease-Time (wurde dem Client mit dem DHCPACK mitgegeben) soll ein Client dem Server mitteilen, daß er die vergebene Konfiguration weiterhin behalten will. Dies erfolgt nun als Unicast-DHCPREQUEST an den betreffenden Server. Dieser kann mit einem DHCPACK mit den gleichen Daten, aber neuer Lease-Time antworten.

Der Client kann allerdings seine Konfiguration noch weiter verwenden, sollte der Server nicht antworten. Er muss dann nach Ablauf von 7/8 der Lease-Time bei *irgendeinem* DHCP-Server im Netz eine Verlängerung beantragen. Sollte auch das nicht funktionieren, muss der Client sein Netzwerkinterface dekonfigurieren und ganz von vorn mit einem DHCPDISCOVER beginnen.

Sicherheit DHCP baut wie auch BOOTP auf UDP und IP auf und ist damit schon inhärent unsicher. Weil DHCP außerdem ja gerade dafür geschaffen wurde, netzsensitive Informationen zu verteilen, eröffnen sich vielfältige Angriffsmöglichkeiten:

Unauthorisierte DHCP-Server sind relativ einfach aufzusetzen. Solche Server könnten dann falsche und potenziell kommunikationsunterbrechende Informationen an Clients senden. Beispiele wären inkorrekte oder doppelte IP-Adressen, falsche Routing-Informationen, falsche DNS-Server-Adressen ...

Böswillige DHCP Clients wiederum könnten sich für legitime Clients ausgeben und Informationen einholen, die für diese bestimmt waren. Wenn Ressourcen dynamisch verteilt werden (z. B. IP-Adressen), könnte solch ein Angreifer alle Ressourcen für sich beanspruchen und dadurch anderen Clients verweigern.

Abhilfe schafft hier die „Authentication for DHCP Messages“ (RFC 3118), die DHCP-Pakete um eine DHCP-Option zur Authentifizierung erweitert.

Fazit DHCP ist eine sinnvolle Erweiterung von BOOTP um variable Lease-Times, zusätzliche Optionen, Relays und der Möglichkeit, mehrere Server im selben Netz zu betreiben. Allerdings erbt es von BOOTP allerdings auch die Nachteile: es wird eine zentrale Instanz benötigt, die ausfallen kann und vor allem bleiben die Sicherheitsimplikationen: unauthorisierte DHCP-Server können einfach dem Netz hinzugefügt werden und falsche Informationen verteilen, böswillige Clients können sich als andere Knoten ausgeben und alle Ressourcen für sich reservieren.

[DHCP-RFC]

PXE

Das Preboot Execution Environment (PXE) ist eine Technologie, die ein netzwerkbaiertes Booten *unabhängig vom Betriebssystem* ermöglicht. Es wurde von 1999 von Intel entwickelt und ist momentan für IA-32 sowie IA-64 verfügbar. Für IA-32 reicht es im Allgemeinen aus, eine mit PXE-Firmware-Erweiterung ausgestatte Netzwerkkarte zu installieren.

Es existiert eine freie Implementierung namens Etherboot, die Client-Funktionalität bereitstellt (z. B. als ROM auf einer Netzwerkkarte). Auf der Serverseite versteht der ISC dhcpd PXE, freie TFTP-Implementierungen existieren ebenfalls.

PXE besteht im Wesentlichen aus drei Kerntechnologien:

- einem einheitlichen Protokoll, daß es einem Client ermöglicht, nach einer Netzwerkadresse und einem Boot-Image zu fragen ? Hier wird eine Kombination von DHCP und TFTP verwendet.
- einer einheitlichen API, die dem Client-Firmware und dem Boot-Image bestimmte Dienste zur Verfügung stellt
- einer standardisierten Art und Weise, wie PXE von der Client-Firmware initiiert wird

Der Netzboot wird dabei im Allgemeinen von einem auf der Netzwerkkarte installierten BIOS angestoßen, oder - wenn dies nicht unterstützt wird - von einem vorher geladenen Firmware-Image (z. B. von Flash-Speicher oder Festplatte).

Funktionsweise

Proxy DHCP Die Firmware des Clients sucht zuerst nach PXE-kompatiblen „Redirection Services“. Das sind spezielle DHCP-Server, die entweder normales und „PXE-Proxy“-DHCP sprechen, oder verschiedene Server, die sich jeweils um normales und „PXE-Proxy“-DHCP kümmern.

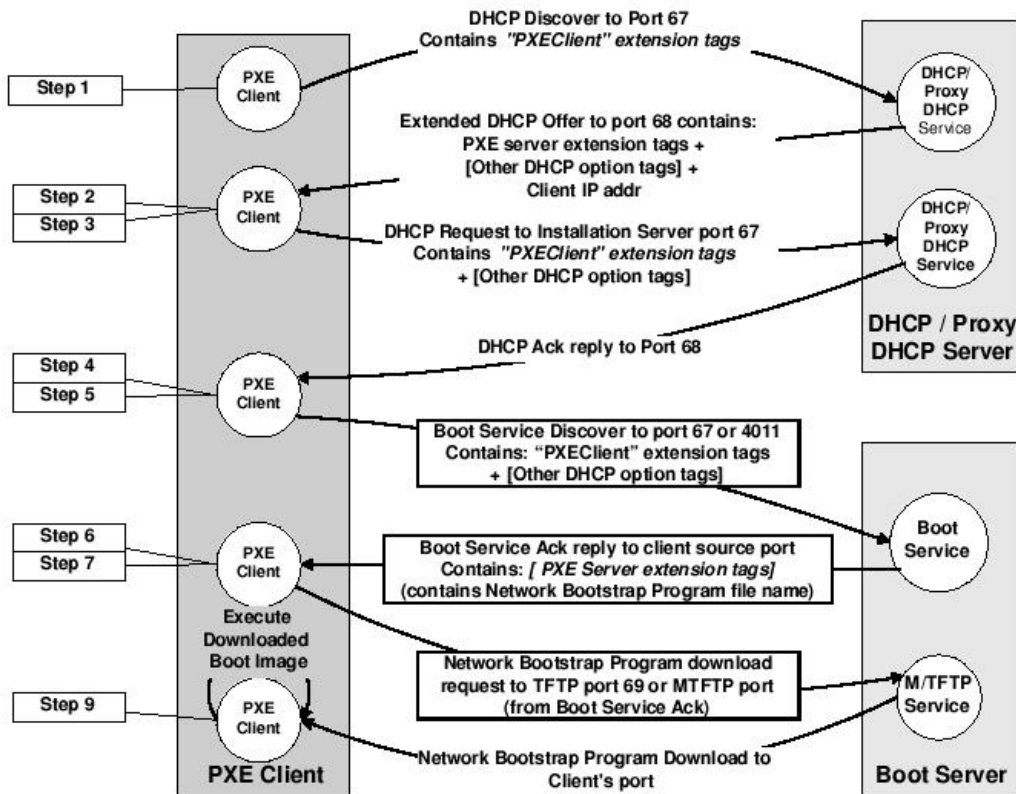
1. Der Client broadcastet also ein mit PXE-Optionen versehenes DHCPDISCOVER an potentielle DHCP-Server.

2. Diese antworten mit einem mit PXE-Optionen versehenen DHCPOFFER-Broadcast. Neben den üblichen DHCP-Optionen (wie IP-Adresse) werden hier nun außerdem noch Informationen über Boot-Server mitgeteilt:

- Liste von IP-Adressen für jeden verfügbaren Boot-Server-Typ
- Kommunikation mit Boot-Server Multicast, Unicast oder Broadcast?

3. Hat der Client nicht schon vorher eine IP-Adresse erhalten, muss die nun erhaltene mit einem DHCPREQUEST an den auserwählten Server bestätigen.

4. Der Server antwortet mit einem DHCPACK.



Boot-Server Auch die Kommunikation mit dem Boot-Server läuft wiederum mittels DHCP ab:

5. Der Client wählt nun automatisch (oder durch einen Benutzer via Bootmenü) einen der angebotenen Boot-Server aus und kontaktiert diesen mit einem um PXE-Optionen erweiterten DHCPREQUEST an Port 4011 oder 67 (im Falle eines Broadcasts). Die Anfrage enthält den Typ des Boot-Servers. 6. Der Boot-Server wiederum antwortet nun mit einem Unicast-DHCPACK, mit PXE-Optionen versehen:

- den Pfad zum einem via TFTP herunterladbaren Boot-Image
- PXE Boot-Server-Typ, auf den der Server antwortete
- soll Multicast TFTP benutzt werden, dessen Konfiguration

7. Der Client lädt das Boot-Image via TFTP oder MFTFTP herunter.

8. Der Server kann außerdem noch eine Checksumme des Boot-Images bereithalten, die der Client aber mit einem erweiterten DHCPREQUEST anfragen muss.

9. Je nachdem ob der Authentizitätstest erfolgreich war, führt der Client das Boot-Image aus.

Sicherheit Es existieren die gleichen Probleme wie schon bei DHCP und BOOTP, die fehlende Möglichkeit der Authentifizierung: PXE sieht keinen Mechanismus vor, zu verhindern, daß ein böswilliger Server wie auch immer geartete Boot-Images verteilt. Andererseits ist es für einen fremden Client möglich, ein Boot-Image zu beziehen, sobald er Zugang zum Netz hat.

[PXE-Draft, PXE-Spec]

IPv4LL - Internet Protocol v4 Link-Local

Das Protokoll IPv4LL (bei Microsoft: APIPA - Automatic Private IP Addressing) dient dazu, daß ein Netzwerkteilnehmer sich in einem IPv4-Netz selbst eine IP-Adresse aus einem für diese Verwendung reservierten Adressraum zuweisen kann. Dies ermöglicht die Kommunikation von Netzwerkgeräten untereinander ohne Konfiguration, auch dann wenn kein DHCP-Server verfügbar ist, wie etwa in Ad-Hoc-Netzen. IPv4LL wurde 2005 in RFC 3927 spezifiziert und ist für die meisten modernen Betriebssysteme verfügbar.

IPv4LL dient der verbindungslokalen Kommunikation von Hosts. Im Sinne von IPv4LL ist eine Gruppe von Netzwerkhosts Link-Local wenn sie die folgenden Bedingungen erfüllen:

1. Wenn jeder Host A Broad-, Multi-, oder Unicast-Pakete zu Host B schicken kann und deren Link-Layer-Transportdaten (Payload) unverändert bei Host B ankommen.
2. Wenn ein Broadcast auf dieser Verbindung, der von einem Host A versandt wird, von allen Hosts aus der Gruppe empfangen werden kann.

Zur Verwendung für Link-Lokale Kommunikation wurden von der Internet Authority for Assigned Names and Numbers (IANA) das IPv4-Präfix 169.254/16 reserviert. Diese Adressen sind, wie das LL für Link-Local schon impliziert, nur auf dem lokalen Link gültig, d.h. sie werden nicht geroutet (ähnlich etwa zu den 192.168/16 und 10/24 Präfixen). Die Adressen des 169.254/16 sollen nur zur Autokonfiguration verwendet werden und nicht manuell oder von einem DHCP-Server vergeben werden.

IPv4LL Adressen dürfen nur dann vergeben bzw. automatisch konfiguriert werden, wenn der Host keine Adresse außerhalb des IPv4-Präfix 169.254/16 besitzt. Sobald ein Host eine routebare Adresse besitzt, sollen keine 169.254/16-Adressen mehr vergeben werden.

Funktionsweise Wenn ein Host eine IPv4LL-Adresse verwenden möchte, wählt er zufällig (mit einem Pseudozufallszahlengenerator gleichmäßiger Verteilung) eine Adresse aus dem 169.254/16-Adressraum aus. Dabei sind die ersten und letzten 254 Adressen reserviert und dürfen nicht ausgewählt werden. Dabei müssen die Zufallszahlen so gewählt werden, dass zwei Hosts nach Möglichkeit nicht dieselbe IP-Adresse wählen. Dazu bietet sich die Verwendung der MAC-Adresse als Seed an.

Nach der Auswahl der IP-Adresse wird eine ARP-Anfrage nach der Adresse gestartet, um zu bestimmen, ob die Adresse schon in Verwendung ist. Ist diese Anfrage negativ, so wählt der Host diese Adresse, im anderen Fall beginnt er mit der Wahl einer neuen Zufallsadresse.

Nach der erfolgreichen Wahl einer Adresse nimmt der Host die Adresse in Besitz, indem er ein Gratuitous ARP versendet, also einen ARP-Anforderungs-Broadcast verschickt, bei der er seine eigene IP-Adresse als Quell- und Ziel-IP-Adresse einträgt. Dies ist notwendig um die ARP-Caches der anderen Hosts am Link zu leeren.

Bei der Verwendung von IPv4LL-Adressen wird die TTL aller Pakete explizit auf 1 gesetzt, um ein Routen zu verhindern, da die Pakete nur link-lokal sind.

Sicherheit Potenzielle Angreifer könnten auf alle Anfragen zur Duplikatserkennung positiv antworten und so alle Ressourcen, sprich IP-Adressen für sich reservieren.

Die üblichen Spoofing-Probleme existieren natürlich auch: Es ist für einen link-lokalen Angreifer möglich, ARP-Pakete zu senden, die einen Host all seine Verbindungen abbrechen lassen würden.

[IPv4-RFC]

2.1.2 Namensauflösung

Es existieren momentan zwei sehr ähnliche Ansätze, herauszufinden, welcher Netzknoten welchen bestimmten Namen hat:

Apples Multicast DNS (mDNS) wird benutzt (z. B. in OS X ab 10.2) und wird offen vertrieben. Es ist allerdings noch nicht standardisiert. Microsofts Link-Local Multicast Name Resolution (LLMNR) wird noch wenig benutzt, wurde allerdings als („Informational“-) RFC 4795 veröffentlicht.

Beide Protokolle haben nur geringfügige Unterschiede: mDNS erlaubt es Knoten, einen DNS-Namen in der `.local` Domain auszuwählen und diesen mittels einer speziellen Multicast IP-Adresse anzukündigen. LLMNR dagegen erlaubt es Knoten, jeden beliebigen Domain-Namen auszuwählen.

mDNS ist kompatibel zu DNS-SD, das im nächsten Abschnitt vorgestellt wird, LLMNR nicht.

mDNS - Apple

Das Ziel von Multicast DNS ist es, link-lokale DNS-Abfragen ohne die Präsenz eines klassischen Unicast DNS-Servers zu ermöglichen. Zusätzlich weist mDNS einen Teil des DNS-Namensraums exklusiv¹ zur Verwendung für link-lokale mDNS-Abfragen zu, aus

¹Aufgrund von internen Uneinigigkeiten wurde dies von der IETF nicht zum Standard erhoben. Das `.local`-Suffix wird aber von in der Praxis verwendet, wie spezifiziert.

dem Namen ohne Registrierung oder Gebühr verwendet werden können. Die Hauptigenschaften von Multicast DNS sind:

1. Es benötigt wenig bis keine Administration.
2. Es funktioniert ohne zusätzliche Infrastruktur wie z.B. DHCP- oder DNS-Server.
3. Es funktioniert auch, wenn Teile der Infrastruktur ausfallen.

Funktionsweise mDNS setzt auf dem bestehenden DNS-Protokoll auf. Die verwendeten Pakete, ihre Struktur und die allgemeine Funktionsweise von DNS bleiben wie in den RFCs 1034 und 1035 spezifiziert erhalten, mit dem einzigen Unterschied, daß DNS-Pakete an Port 5353 gesandt werden. Dies trennt den mDNS-Verkehr von regulärem DNS-Verkehr. Die mDNS-Spezifikation beschreibt lediglich die Voraussetzungen dafür, daß DNS-Anfragen in einem IP-Netz an eine Multicast-Adresse gesandt werden können. Da mDNS über IP-Multicast funktioniert, kann es sowohl mit IPv4 als auch mit IPv6 verwendet werden.

Multicast-Anfragen Eine DNS-Anfrage mit der Endung `.local` muss an die mDNS-Multicastadresse 224.0.0.251 bzw. FF02::FB gesendet werden, also an die IP-Multicast-Adresse, die alle Teilnehmer des lokalen Links erreicht. Zusätzlich können alle DNS-Anfragen, die nicht auf `.local` enden, auch an diese Adressen gesendet werden, sofern kein konventioneller DNS-Server verfügbar ist. Die ist notwendig, um mDNS und reguläres DNS parallel betreiben zu können. So ist es möglich, dass ein Client die Namen auf dem lokalen Link über mDNS auflöst, während er gleichzeitig Namen außerhalb dieses Raums (etwa aus dem Internet) über einen klassischen DNS-Server auflöst. Clients wählen sich selbstständig Namen aus diesem „flachen“ Namensraum aus. Herausgestellt werden sollte hier auch, dass zwar für die meisten Anwendungen *empfohlen* ist, für Eindeutigkeit von Namen zu sorgen, daß ein Client also prüft, ob ein Name schon vergeben ist und in einem solchen Fall einen neuen wählt. Dieses Verhalten wird jedoch *nicht verlangt*, um Anwendungen wie z. B. „Load-Balancing“ oder ähnliches zu ermöglichen.

Unicast-Anfragen Das erste Bit des Class-Field eines DNS-Pakets wird von mDNS als Unicast-Response-Bit betrachtet. Wenn dieses Bit bei einer Anfrage gesetzt ist, dann zeigt ein Client an, dass er auch Unicast-Antworten akzeptiert. Das Bit sollte insbesondere dann gesetzt werden, wenn es für die anderen Hosts am lokalen Link nicht notwendig ist, die Antwort mitzubekommen.

Der Responder sollte in diesem Fall via Unicast antworten, es sei denn, die Antwort wurde seit längerem nicht mehr über Multicast verbreitet, dann soll die Antwort über Multicast gegeben werden, um Caches der Clients auf dem aktuellen Stand zu halten. Als Richtwert wird hier ein Viertel der TTL des DNS-Eintrags angegeben.

Antworten Die Antworten sind gewöhnliche (RFC 1035 und 1034 konforme) DNS-Antworten, die ebenfalls an die Adresse der Multicast-Gruppe für den lokalen Link (224.0.0.251) an Port 5353 gesandt werden, und so von allen mDNS-Respondern² im lokalen Netz empfangen werden. Diese Eigenschaft wird auch für Caching verwendet. Antworten, die nicht aus dem lokalen Subnetz stammen, werden ignoriert.

Da alle anderen mDNS-Clients die Anfrage ebenfalls hören, ist es jedem Client möglich, einen Cache nicht nur über die eigenen, sondern auch über fremde Anfragen zu führen. Dies reduziert die Menge an Kommunikation, die nötig ist, um eine mDNS Anfrage aufzulösen stark.

Im Unterschied zum klassischen DNS-System, bei dem jeder DNS-Server mit allen ihm bekannten Informationen auf eine Anfrage antwortet, auch wenn er nicht *authoritative* für diese Angaben ist, darf ein mDNS-Responder nur mit den Informationen antworten, für die er auch *authoritative* ist. Ein mDNS-Responder antwortet nur, sofern er eine gültige Antwort hat. Fehler-Antworten werden nicht gegeben, so dass ein Client nur über das Ausbleiben von Antworten feststellen kann, dass kein Eintrag für seine Anfrage existiert.

Um Kollisionsfreiheit in Shared-Medium-Netzen (wie z.B. 802.11 WLAN) zu garantieren, verlangt die mDNS-Spezifikation von mDNS-Respondern, Antworten erst nach einer zufälligen künstlichen Verzögerung von 0-500 ms zu senden, außer wenn der Responder glaubt, der einzige Antwortende zu sein.

Reverse Mapping Jede DNS-Anfrage die mit `254.169.in-addr.arpa` endet, muss an die mDNS-Multicast-Adresse gesendet werden. Ebenso müssen in einem IPv6-Netz Anfragen, die auf `8.e.f.ip6.arpa` enden an `FF02::FB` gesandt werden. Darüber ist Reverse-Mapping analog zum klassischen DNS-Reverse-Mapping möglich.

Ankündigung und Nameskollisionsvermeidung Wenn ein Host mit einem neuen Netz verbunden wurde, aus dem Standby-Modus erwacht ist, angeschaltet wurde oder sich der Status des Links eines oder mehrerer Interfaces auf andere Weise geändert hat, muss der Client Schritte zur Kollisionsvermeidung und eine Ankündigung seines Namens durchführen.

Dazu sendet er zunächst eine Anfrage nach dem von ihm gewünschten Namen mit dem DNS-Query-Type `T_ANY` (Byte-Wert 255), um Antworten für alle Eintragungen mit diesem Namen zu provozieren. So ist dem Host möglich, exklusiven Besitz eines Eintrages festzustellen, bevor er diesen verwendet.

Diese Abfrage wird zufällig bis zu 250ms verzögert, um Kollisionen zu vermeiden, wenn mehrere Geräte gleichzeitig angeschaltet bzw. aufgeweckt werden. Dann muss der Client

²Da mDNS keine klassische Client/Server-Kommunikationsstruktur verwendet, ist der Name mDNS-Server/Client nicht passend. mDNS verwendet den Begriff des Clients für denjenigen, der Anfragen stellt und den des Responders für den Antwortenden.

drei Anfragen im Abstand von 250ms absenden. Sollten 250ms nach der dritten Anfrage noch keine Antworten eingetroffen sein, so soll der Client den Namen als kollisionsfrei betrachten und zur Ankündigung übergehen. Ansonsten muss er einen anderen Namen wählen und von vorne beginnen.

Zum Ankündigen sendet der Client eine Antwort mit dem höchstwertigen Bit der RRCLASS auf 1, um eine Cache-Leerung auszulösen und sich so anzukündigen. Ein Client muss zwischen zwei und acht Ankündigungen senden.

Die Spezifikation schreibt, z. B. im Gegensatz zu LLMNR vor, keine wiederholten kontinuierlichen Ankündigungen ohne Grund zu senden, um das Netzwerk nicht zu belasten. Ankündigungen werden auch dann verschickt, wenn sich ein RR.RECORD ändert.

Sicherheit Da IPv4LL nur link-lokal funktioniert, muss sich ein potentieller Angreifer am selben Linksegment wie das Opfer befinden. Dies schließt einen großen Teil klassischer DNS-Angriffe aus. Auch der andere Cache-Algorithmus, in Kombination damit, daß jeder auf seine eigenen Hostnamen autoritativ antworten kann, stellt eine gewisse Robustheit dar. Dies kann allerdings Spoofing-Angriffe nicht verhindern.

mDNS unterstützt DNSSEC für die Signierung von DNS-Einträgen mittels Public-Key-Kryptografieverfahren. Allerdings erfordert dies eine vorherige Verteilung von öffentlichen Schlüsseln auf den Clients, was dem Autokonfigurationsgedanken doch sehr im Wege steht.

[mDNS-Draft]

LLMNR - Microsoft

LLMNR, die „Link-local Multicast Name Resolution“ wird im RFC 4795 (Informational) beschrieben. Es ermöglicht es IPv6- und IPv4-Hosts, die Hostnamen von Computern im gleichen Netzwerk ohne einen DNS-Server oder einen DNS-Client aufzulösen. LLMNR wurde von Microsoft entwickelt, um Namensauflösung ohne DNS-Server in IPv6-Netzen analog zu NetBIOS-over-TCP/IP-Namensauflösungen zu ermöglichen, bzw. das NetBIOS-over-TCP/IP-Protokoll in IPv4-Netzen durch LLMNR zu ersetzen.

Funktionsweise LLMNR verwendet dazu Multicast und ist sowohl in IPv4- als auch in IPv6-Netzen einsetzbar.

LLMNR-Nachrichten nutzen ein Format, das dem von DNS-Nachrichten nach RFC 1035 entspricht, sendet diese jedoch an einen anderen Port. LLMNR-Name-Query Request-Nachrichten werden unter der Windows Vista Implementation an den UDP-Port 5355 versendet. Die Spezifikation beschreibt LLMNR-Anfragen sowohl über TCP- als auch über UDP-Verbindungen, analog zu DNS-Anfragen. LLMNR-Name-Query Response-Nachrichten werden über UDP-Port 5355 gesendet. LLMNR- und DNS-Stacks verwenden einen getrennten Cache.

Eine LLMNR-Anfrage wird unter IPv6 an die Link-Lokale-Multicast-Adresse FF02::1:3 gesendet, während sie bei IPv4 an 224.0.0.252 gesandt werden.

Bei LLMNR werden die Anfragen über Multicast, die Antworten jedoch über Unicast gesendet, wobei jeder Host, im Gegensatz zum DNS-Verfahren, nur auf Anfragen nach seinem eigenen Hostnamen antwortet. Es gibt keinen abgeteilten DNS-Namensraum wie z. B. bei mDNS.

Paketformat Ein LLMNR-Paket ist wie folgt aufgebaut:

0	8	16	24
LLMNR Header (12 Byte)			
Anfrage-Einträge (variabel)			
Antwort-Einträge (variabel)			
Authority-Einträge (variabel)			
zusätzliche Einträge (variabel)			

Der Header wiederum hat folgende Struktur:

0	8	16	24
Transaktions-ID		Flags und Optionen	
Anzahl Anfrage-Einträge		Anzahl Antwort-Einträge	
Anzahl Authority-Einträge		Anzahl zusätzliche Einträge	

- **Transaktions ID:** Analog zu DNS wird bei LLMNR eine zwei Byte lange Transaktions-ID verwendet, mit deren Hilfe Anfragen den entsprechenden Antworten zugeordnet werden können.
- **FLAGS UND OPTIONEN:**
 - *QR* (1 Bit): Paket ist Anfrage (Query QR=0) oder Antwort (Response QU=1)
 - *Opcode* (4 Bit): Zeigt die Art der Abfrage an. Ist bei einer gewöhnlichen Anfrage/Antwort 0
 - *C-Flag* (1 Bit): Ein Wert von 1 zeigt einen Namenskonflikt an, sobald der Name in einem Subnetz nicht eindeutig ist. Sonst wird der Wert auf 0 gesetzt. Anfragen, bei denen das C-Flag 1 ist werden nicht beantwortet.
 - *TC-Flag* (1 Bit): Dieses Truncation-Flag wird gesetzt, wenn die Größe der Antwort die Maximalgröße des Paktes übersteigt.
 - *T-Flag* (1 Bit): Das Tentative-Flag wird auf 1 gesetzt, wenn der antwortende Host den Namen verwendet, aber nicht feststellen konnte, ob der Name eindeutig ist.
 - *RCODE-Field* (4 Bit): Der Antwortcode der Antwort, analog zu DNS, wobei das Feld bei gültigen Antworten auf 0 gesetzt wird. Nicht autorisierte Antworten, bei DNS RCODE 3, gibt es bei LLMNR nicht, da jeder Host nur für sich selbst antwortet.

- Zusätzlich wird im Header die Größe der Antwortdaten übertragen.

Die Maximalgröße einer LLMNR-Nachricht richtet sich nach der maximalen Paketgröße des verwendeten Transportprotokolls (IPv6 UDP, IPv4 UDP bzw. TCP).

Ein LLMNR-Paket hat folgende Felder gesetzt:

- Quelladresse bei IPv4 auf die Unicast-Adresse des sendenden Interface und bei IPv6 die Uni- oder Multicast-Adresse des sendenden Interface.
- Zieladresse auf FF02::1:3 (IPv6) bzw. 224.0.0.252 (IPv4), also jeweils die Multicast-Adresse des lokalen Links.
- TTL-Feld bzw. das Hop-Count Feld bei IPv4 respektive v6 wird auf 1 gesetzt, da nur auf dem lokalen Link kommuniziert wird..

Eindeutigkeitsprüfung und Namensauflösung Zur Eindeutigkeitsprüfung des eigenen Hostnamens sendet ein Host eine Anfrage nach seinem Hostnamen über LLMNR. Wenn er eine Antwort mit dem C-Flag = 1 erhält, so gibt es einen Namenkonflikt. Daraufhin antwortet der Host nicht mehr auf Anfragen nach diesem Namen, sendet jedoch alle 15 Minuten eine neue Anfrage nach seinem Hostnamen, um festzustellen, ob der Host, der den Konflikt ausgelöst hat, nicht mehr vorhanden ist und er diesen wieder verwenden kann.

Ein Windows Vista Computer konvertiert Anfragen nach `hostname.local` in Anfragen nach `hostname` und versucht diese über LLMNR aufzulösen. Daher ist der `.local`-Namensraum mit Vista weder für DNS noch für mDNS verwendbar.

Sicherheit LLMNR wird link-lokal verwendet. Daher sind viele Attacken, die auf DNS-Systeme möglich sind, nicht möglich. Um ein LLMNR System anzugreifen, muss der Angreifer sich am selben Linksegment befinden. Eine Absicherung der Authentizität der Einträge kann laut Spezifikation mittels DNSSEC oder TSIG sichergestellt werden. Allerdings bedeutet dies ein vorheriges Verteilen von Schlüsseln, was dem Autokonfigurationsgedanken in gewisser Weise entgegen steht.

[LLMNR-MSTechnet, mDNS-Draft, LLMNR-RFC]

2.1.3 Diensterkennung

DNS-SD - Apple

„Domain Name System based Service Discovery“ ist eine Erweiterung des DNS Systems um die Fähigkeit, Dienstanmeldung und -suche in Netzwerken durchzuführen. Eines der Designziele von DNS-SD war die einfache Implementierung und daraus entstehende

simple Integrierbarkeit von DNS-SD in bestehende DNS-Systeme. Dabei ist DNS-SD eine Ergänzung zu den bestehenden Möglichkeiten von DNS und versucht nicht, diese zu ersetzen.

DNS-SD wurde entwickelt, um die drei Kernfunktionen eines Diensterkennungsprotokolls zu ermöglichen:

1. Die Fähigkeit nach einem bestimmten Dienst-Typ zu suchen und eine Liste von Dienst-Instanzen zu finden.
2. Durch eine Dienst-Instanz die zur Verwendung des Dienstes nötigen Informationen (IP-Adressen, Portnummern, etc.) zu erhalten
3. Die Instanznamen sollten langlebig sein, selbst wenn sich die zugrundeliegenden Dienste ändern (etwa wenn ein Drucker seine IP-Adresse wechselt).

DNS-SD sieht bewusst keine Änderung am DNS-Protokoll vor, sondern beschreibt lediglich eine Konvention, wie sich bestimmte bereits definierte DNS-Ressourcen strukturieren lassen, um DNS-SD zu ermöglichen. Es ähnelt in dieser Hinsicht dem Sender Policy Framework SPF das zur Überprüfung der Authorisierung von SMTP-Server zum Senden von E-Mail verwendet wird.

Dieses Design ermöglicht es, DNS-SD mit der heute weit verbreiteten klassischen Unicast-DNS Infrastruktur oder alternativ mit der Multicast-DNS Infrastruktur zu verwenden.

Ein weiteres Designziel bei DNS-SD war eine einfach mögliche Implementierung, um eine weite Verbreitung zu ermöglichen. Aus diesem Grund sind auch alternative Implementierungen erwünscht, wie sie etwa das Avahi-Projekt entwickelt.

Funktionsweise DNS-SD verwendet DNS-Records der Typen TXT, PTR und SRV. SRV-Records haben gegenüber den üblichen A (bzw. AAAA bei IPv6) Einträgen den Vorteil, dass Zusatzinformationen (wie etwa die Portnummer) in der Antwort mitgeliefert werden können.

Der abfragende Client führt eine DNS-Abfrage des Typs PTR nach einem auf spezielle Weise strukturierten DNS-Namen aus. Die Struktur einer solchen Abfrage ist

```
<Dienstname>.<Diensttyp>.<Domain>
```

Wobei, wie bei SRV-Records üblich, <Diensttyp> entweder `_udp` oder `_tcp` ist, je nach vom Dienst verwendeten Protokoll³. <Dienstname> ist der eindeutige Name des Dienstes (z.B. `_ipp` für das Internet Printing Protocol). Um die Eindeutigkeit der Namen sicherzustellen, sollen diese DNS SRV Service Types bei `dns-sd.org` registriert werden, wobei sie sich nicht mit den bei IANA registrierten „assigned port names and numbers“

³SRV-Records werden in RFC 2782 spezifiziert.

überschneiden dürfen, es sei denn, für diesen Dienst ist dort dieser Name registriert. Dienstnamen unterliegen den Zeichensatzbeschränkungen von Hostnamen. Domain ist die DNS-Domain der Dienstanfrage (z.B. `example.com`). Diese kann `.local` sein, was eine link-lokale Multicast DNS-Anfrage impliziert⁴.

Eine solche Dienstanfrage lässt sich wie ein Verzeichnisspfad eines Dateisystems lesen, wobei DNS-Einträge natürlich von rechts nach links gelesen werden müssen. Die Anfrage nach der Liste aller Dienste für `_http._tcp.example.com` ließe sich also verstehen als `/com/example/_tcp/_http`.

Das Ergebnis der Anfrage wird auf ähnliche Weise ausgedrückt. Das Ergebnis einer PTR-Anfrage der obigen Form sind null oder mehr PTR-Records der Dienstinstanzform (Service Instance Name):

`<Instanz>.<Dienst>.<Domain>`

`<Instanz>` ist hierbei ein DNS-Label in UTF-8 enkodierter Form⁵, also der Name der Instanz in menschenlesbarer Form, wie er etwa in einem Benutzerinterface präsentiert wird.

Das DNS-SD Protokoll sieht vor, dass ein Client zunächst nach den PTR-Records eines ihm bekannten Dienstes fragt. Wenn er einen der zurückgegebenen Dienste verwenden möchte (und das kann durchaus einige Zeit später sein) präsentiert er dem Benutzer eine Auswahlliste aller verfügbaren Instanzen des Dienstes und fragt nach allen Einträgen des ausgewählten Dienstes. Dies sind der SRV-Record, alle TXT-Records sowie zusätzlich die A- und AAAA-Records des Hosts, um die Auflösung zu beschleunigen.

Zusätzlich muss zu jedem SRV-Record noch mindestens ein TXT-Records mit demselben Namen übertragen werden. Dort werden, falls die Anwendung mehr Informationen benötigt als IP-Adresse und Portnummer, zusätzliche Informationen in einer strukturierten Form abgelegt. Es können mehrere TXT-Records zurückgegeben werden. Leere TXT-Records enthalten nur ein einzelnes NULL-Byte, da leere TXT-Einträge nicht erlaubt sind (siehe RFC 1035). TXT-Records, die Daten enthalten, halten diese in einer Schlüssel=Wert Form bereit. Ein Beispiel für die Verwendung von TXT-Records ist z. B. das LPR-Druckprotokoll, bei dem der Warteschlangenname als TXT-Records encodiert wird.

Beispiele Mit Hilfe des UNIX-Tools `dig` lassen sich DNS-SD-Abfragen testen. Das folgende Kommando listet alle `_http._tcp`-Ressourcen, also Webseiten auf.

⁴Wobei dies aufgrund interner Differenzen in der IETF bisher nicht zum Standard erklärt wurde, daher kann es sich bei `.local` auch um eine (nicht global gültige) traditionelle Unicast-DNS Domain handeln.

⁵Nach RFC 3629. Das DNS-Protokoll unterstützt im Gegensatz zu üblichen Annahmen auch Nicht-ASCII-Einträge. Das Internet-DNS System beschränkt sich jedoch (mittels Punycode) auf ASCII-Symbole, da diese sich in einer Textkonsole leichter eingeben lassen. Da die Instanznamen bei DNS-SD niemals von Hand eingegeben werden sollen, spielt diese Beschränkung dort keine Rolle.

```
dig -t ptr _http._tcp.example.com
```

Bei Verwendung zusammen mit mDNS muss die Abfrage folgende Struktur haben, da die Anfrage an Port 5353 der link-lokalen Multicastadresse gesendet wird.

```
dig -p 5353 -t ptr @224.0.0.251 _http._tcp.local
```

Eine typische Antwort wäre:

```
_http._tcp.local. 10 IN PTR LaserJet 6L._http._tcp.local.  
LaserJet 6L._http._tcp.local. 10 IN TXT "path=/"  
LaserJet 6L._http._tcp.local. 10 IN SRV 0 0 80 laserdrucker.local.  
laserdrucker.local. 10 IN A 192.168.1.111  
_http._tcp.local. 10 IN PTR Web Server on pyrite._http._tcp.local.
```

Dank DNS kann der antwortende Server alle Informationen, die ein Client benötigt, um die Resource zu verwenden, gleich mit bereit stellen, so dass die Antwort nicht nur eine Liste aller Dienste des _http._tcp-Typs umfasst, sondern auch gleich deren SRV- und TXT-Einträge, sowie den A-Eintrag mit der IP-Adresse.

Trotzdem sollte ein Client vor dem Verwenden einer Ressource eine Abfrage dieser Resource vornehmen.

```
dig -t srv 'LaserJet 6L._http._tcp.example.com'
```

bzw. analog bei mDNS

```
dig -p 5353 -t srv @224.0.0.251 'LaserJet 6L._http._tcp.local.'
```

wobei die Antwort typischerweise die folgende Form hätte

```
LaserJet L._http._tcp.local. 10 IN SRV 0 0 80 laserdrucker.local.  
laserdrucker.local. 10 IN A 192.168.1.111
```

Sicherheit Für DNS-SD gelten die für mDNS gemachten Bemerkungen zur Sicherheit. Insbesondere Spoofing und Masquerading machen hier Probleme aus. Wie bei mDNS wäre die Benutzung von DNSSEC eine Möglichkeit.

[DNSSD-AppleDevCon, DNSSD-Draft, DNSSD-LinuxMag]

SSDP - Microsoft

Das Simple Service Discovery Protocol SSDP wird zur Suche nach UPnP-kompatiblen Geräten im Netzwerk verwendet. Es wurde um 1999 von Microsoft und Hewlett Packard entwickelt und ist ein fester Bestandteil von Universal Plug and Play.

Funktionsweise Hat ein UPnP-Gerät eine IP-Adresse erhalten, muss seine Existenz eventuellen Clients mitteilen. Dies erfolgt über SSDP: das UPnP-Gerät sendet eine SSDP-Nachricht an die Multicast-Adresse 239.255.255.250:1900.

Umgekehrt können auch Clients mittels SSDP nach UPnP-Geräten suchen. Die Anfrage geht wiederum mit UDP an die Multicast-Adresse 239.255.255.250:1900. UPnP-Geräte, die den angefragten Dienst anbieten, antworten mit Unicast an den Fragesteller.

Die Nachricht enthält dabei jeweils nur die wichtigsten Angaben über das Gerät und seine Dienste, wie z. B. den Gerätenamen, Gerätetyp und eine URL zur genauen Beschreibung des Gerätes.

Paketformat Ein SSDP-Paket ist im Grunde ein HTTP-Request mit der Methode NOTIFY, wobei der HTTP-Body leer bleibt, im Header jedoch UPnP-spezifische Attribute gesetzt werden, z. B. :

- NTS (Notification Sub Type): „ssdp:alive“ zum Anmelden oder „ssdp:byebye“ zum Abmelden eines Geräts.
- NT (Notification Type): Eigenschaft des Geräts.
- USN (Unique Service Name): eindeutige ID des Geräts.
- LOCATION: URL, die auf eine detailliertere Beschreibung verweist.

Ein Beispiel:

```
NOTIFY * HTTP/1.1
SERVER: Linux/2.6.15.2 UPnP/1.0 Mediaserver/1.0
CACHE-CONTROL: max-age=1800
LOCATION: http://192.168.0.10:8080/description.xml
NTS: ssdp:alive
NT: urn:schemas-upnp-org:service:ConnectionManager:1
USN:uuid:550e8400-e29b-11d4-a716-446655440000::
urn:schemas-upnp-org:service:ConnectionManager:1
HOST: 239.255.255.250:1900
```

Sicherheit Auch hier gibt es wieder Probleme mit potenziellem Spoofing und Masquerading: Geräte können falsche Angaben machen oder sich als andere schon existierende Geräte ausgeben und so deren Funktion im Netz stören.

[SSDP-Draft, SSDP-Wikipedia]

SLP - IETF

Wie die anderen schon erwähnten Protokolle dient auch das Service Location Protocol SLP dem Auffinden von Diensten in einem TCP/IP-Netzwerk. Es existieren mittlerweile drei Versionen, definiert ab 1999 in den RFCs 2608, 2609, 2614, 2165, 3059 und 3082.

Funktionsweise SLP teilt Hosts in drei verschiedene Klassen ein:

1. Verzeichnisagenten (directory agents),
2. Dienstagenten (service agents)
3. Benutzeragenten (user agents).

Will ein Host einen Dienst anbieten, so fungiert er als „service agent“ und registriert sein Dienstangebot bei einem „directory agent“. Beide kommunizieren von nun an in festgelegten Zeitintervallen, ob der Dienst noch verfügbar ist oder es eventuell zu einer Adressänderung oder ähnlichem gekommen ist.

Will ein Host einen Dienst nutzen, so fungiert er als „user agent“, der beim „directory agent“ anfragt, ob der gewünschte Dienst von einem oder mehreren „service agents“ angeboten wird.

Ist ein „directory agent“ vorhanden, wird die Kommunikation hauptsächlich über Unicast ablaufen. Kennt ein Host jedoch die Adresse des „directory agents“ noch nicht, muss er diesen erst über eine spezielle Multicast-Adresse erfragen.

Desweiteren sieht SLP ein Szenario für kleine Netzwerke ohne „directory agents“ vor: auch hier stellen „user agents“ Anfragen an eine spezielle Multicast-Adresse, an der „service agents“ lauschen.

Standardmäßig verwendet SLP UDP, für größere Nachrichten kann auch TCP verwendet werden.

Adressierung Ein Dienst wiederum ist durch eine spezielle URL adressierbar, z. B.

```
service:printer:lpr://myprinter/myqueue
```

Hier wird ein eine Queue „myqueue“ auf dem Drucker „myprinter“ adressiert, der das Protokoll „lpr“ unterstützt. `service:printer:lpr` spezifiziert hierbei den Dienst-Typ, `service:printer` den sogenannten abstrakten Dienst-Typ. Dadurch ist es möglich, nach Druckern zu suchen, egal, welches Protokoll sie verwenden.

Zusätzlich können Dienste eine beliebige Anzahl an Attributen haben, z. B.

```
(printer-name=Hugo),  
(printer-natural-language-configured=en-us),  
(printer-location=In my home office),  
(printer-document-format-supported=application/postscript),  
(printer-color-supported=false),  
(printer-compression-supported=deflate,gzip)
```

Das Format der URLs und der Attribute ist in RFC 2609 definiert. Mittels dieses Adressierungsschemas ist es möglich, nach bestimmten Diensten zu suchen und Informationen über sie einzuholen:

- Suche nach allen Diensten mit gleichen (abstrakten) Typ
- dabei möglich: Abfrage von Attributen (RFC 3059)
- Abfrage einer Liste aller verfügbaren Dienst-Typen

Verbreitung

- SLP wird in Novell Netware ab Version 5 als Ersatz für das IPX-basierte Service Advertising Protocol SAP verwendet.
- zum Auffinden von Druckern, z. B. verwendet im „Common Unix Printings System“ CUPS
- MacOS X bis Version 10.1 benutzte SLP anstatt Zeroconf (ergo mDNS und DNS-SD)

Sicherheit SLP enthält einen Sicherheitsmechanismus, der auf asymmetrischer Kryptographie basiert und eine Signierung von Dienstangeboten ermöglicht.

Diese Möglichkeit wird allerdings sehr selten benutzt, da sie es erforderlich macht, die öffentlichen Schlüssel *jedes* „service agents“ auf *jedem* „service user“ zu hinterlegen. Das macht den eigentlichen Nutzen von SLP, das Auffinden von Diensten ohne vorhergehende Konfiguration, hinfällig.

[SLP-Wikipedia]

2.2 Neue Welt - IPv6

In diesem Abschnitt sollen Autokonfigurationstechnologien vorgestellt werden, die sich explizit auf IPv6 beziehen.

Da wir davon ausgehen, daß die Leserschaft im Allgemeinen sehr wohl mit IPv4, jedoch weniger mit dem neueren IPv6 vertraut ist, folgt zuerst eine kurze Zusammenfassung der wichtigsten Neuerungen gegenüber IPv4.

Hauptvorteile von IPv6 gegenüber IPv4 sind unter anderem:

- der viel größere Adressraum: Statt 32 Bit stehen 128 Bit zur Verfügung. Das macht NAT überflüssig.

- vereinfachte Struktur des IP-Headers: Dies macht unter anderem effizienteres Routing möglich.
- Mobile IP.
- Multicast statt Broadcast.
- Sicherheitsfunktionen obligatorisch.
- *Möglichkeit zur Autokonfiguration von vornherein integriert.*

Headerformat

Der IPv6-Header ist gegenüber IPv4 mit 20 Byte nun 40 Byte angewachsen, IPv6-Header können außerdem *verkettet* werden, die Headerlänge bleibt also auch bei Angabe von Optionen *konstant*.



- **Version:** bei IPv6 immer 0x6.
- **Traffic Class:** gleiche Bedeutung wie Type-of-Service (IPv4), auch bezeichnet als „Differentiated Services“-Feld
- **Flow Label:** QoS *auf Netzebene* (IPv4: Analyse der TCP/UDP-Header...)
- **Payload Length:** Anzahl der Nutzdaten-Bytes, eventuelle optionale IPv6-Header mitgezählt
- **Next Header:** Typ des im Paket nachfolgenden Headers
 - z. B. TCP, ICMP etc.

- aber auch IPv6-eigene: Hop-by-Hop-Options, Routing, Fragment, Resource Reservation ...
- **Hop Limit:** wie IPv4-TTL
- **Adressen:** jeweils 128bit:
 - Source: immer eine Unicast-Adresse
 - Destination: Unicast, Multicast oder Anycast

Es ist keine Header-Prüfsumme vorgesehen! Dies wird dadurch begründet, daß Layer2-Protokolle (Ethernet, ATM, PPP etc.) schon durch eigene Prüfsummen eine zuverlässige Fehlererkennung besitzen. Die IPv6-Nutzdaten dagegen müssen jedoch eine Prüfsumme haben, dies wird bewerkstelligt, indem in die Prüfsumme der höheren Protokollschicht *einige* (unveränderliche) Felder des IPv6-Headers mit einbezogen werden. Dieser sogenannte IPv6-Pseudo-Header enthält Source Address, Destination Address, Next Header und Payload Length.

Weiterhin fällt auf, daß im Gegensatz zu IPv4 keine Fragmentierungsfelder vorhanden sind. Passt ein IP-Paket nicht in die vom Link-Layer vorgegebene MTU, muss es fragmentiert werden. Da dies jedoch einen Overhead und damit ineffiziente Datenübertragung bedeutet, wird versucht, Fragmentierung zu vermeiden. Sollte doch fragmentiert werden müssen, kann das in einem zusätzlichen optionalen IPv6-Header angegeben werden. Bei IPv4 dagegen enthält *jeder* Header Fragmentierungsfelder.

Adressierung

Es gibt drei Typen von IPv6-Adressen: Unicast, Multicast (beinhaltet IPv4-Broadcast) und Anycast (wie Multicast, allerdings nur ein Mitglied der adressierten Gruppe).

IPv6-Adressen werden normalerweise in 8 Blöcken à 16 Bit in Hexadezimaldarstellung, abgetrennt durch Doppelpunkte notiert, beispielsweise

1234 : ABCD : DEAD : BEEF : DCBA : 4321 : 5678 : AF FE

Zur Abkürzung darf bei der Notation einer IPv6-Adresse *maximal eine* Folge von Nullen weggelassen werden.

Adresspräfixe Die höchstwertigsten Bits einer IPv6-Adresse dienen zur Kennzeichnung des Netzes (vergleichbar mit CIDR bei IPv4). Die wichtigsten vordefinierten Adressbereiche sind:

Präfix binär	Präfix hexadezimal	Namensbereich
001	20 ... 3F	aggregierbare globale Unicast-Adressen
1111 1110 10	FE80 ... FEBF	verbindungslokale Adressen (link-local)
1111 1110 11	FEC0 ... FEFF	netzlokale Adressen (site-local)
1111 1111	FF	Multicast-Adressen

Site-local bedeutet hier: am gleichen (privaten) Netz, z. B. einer Firma oder Hochschule, *link-local* dagegen, daß Knoten sich im gleichen Layer-2 Adressraum befinden, z. B. der gleichen Ethernet-Broadcast-Domain.

Multicast Bei IPv6 gibt es keine Broadcast-Adressen mehr, stattdessen wurde IPv6-Multicast um etliche spezielle Multicast-Adressen für verschiedenste Zwecke erweitert, z. B. das Auffinden eines Routers oder die Ermittlung einer Link-Layer-Adresse. Der Vorteil besteht darin, daß nicht alle Knoten des Broadcast-Adressraums adressiert werden, sondern nur Mitglieder einer spezifischen Multicast-Gruppe.

Eine Multicast-Nachricht hat folgende Struktur:



- **Flags:** 0 0 0 T: T = 0 ... dauerhaft festgelegt, T = 1 ... transient, nicht dauerhaft festgelegt
- **Scope:** Gültigkeitsbereich
- **Group ID:** kennzeichnet die jeweilige Gruppe innerhalb des jeweiligen Gültigkeitsbereiches

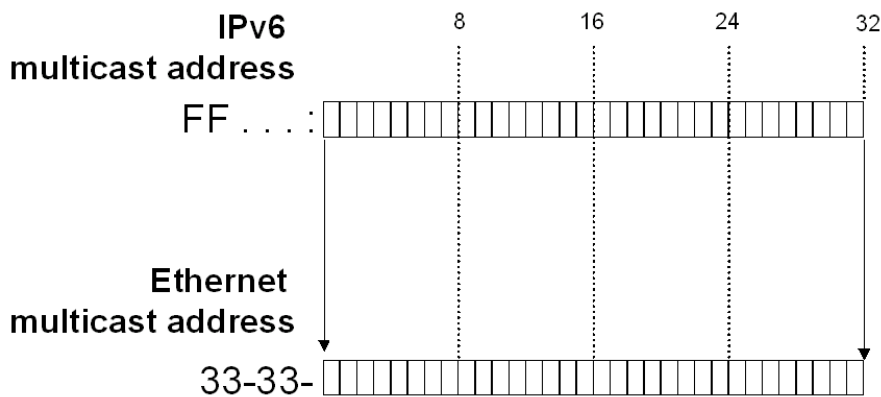
Eine Multicast-Adresse kann nun für folgende Gültigkeitsbereiche gelten:

0	reserviert
1	node-local: gilt nur für diesen Knoten, ggf. für alle Interfaces
2	link-local: gilt für alle Knoten an diesem Link-Segment
5	site-local: gilt für alle Knoten an diesem Netz
8	organization-local: gilt für alle Knoten dieser Organisation
E	global: gilt für alle Knoten im Internet
F	reserviert

Für die Autokonfiguration interessante vordefinierte Multicast-Adressen sind nun diejenigen mit link-local Scope, also FF02-Präfix:

FF02::1	alle Knoten
FF02::2	alle Router
FF02::5	alle OSPF-Router
FF02::9	alle RIP-Router
FF02::1:2	alle DHCP-Agenten
FF02::1:FFxx:xxxx	Solicited-Node-Adresse: Zieladresse zum Finden der MAC-Adresse eines Nachbarknotens mit IP-Adresse yyyy:yyyy:yyyy:yyyy:yyyy:yyxx:xxxx

IPv6-Multicast-Adressen müssen nun irgendwie auf Layer-2-Multicast-Adressen abgebildet werden. Für MAC-Multicast-Adressen werden einfach die unteren 32bit einer IPv6-Multicast-Adresse mit dem Präfix 0x3333 versehen:



ICMP für IPv6

ICMP wurde für IPv6 wesentlich erweitert. Wie bei IPv4 kümmert sich ICMP um Erreichbarkeitstests, Fehlermeldungen und Steuerungsfunktionen (Zeit, Router-Vorschlag). Außerdem übernimmt es alle Funktionen, die bei IPv4 noch in IGMP und ARP realisiert waren. *In IPv6 existieren weder ARP noch IGMP.*

[IPv6-Heise, IPv6-Wiese]

2.2.1 Adresskonfiguration

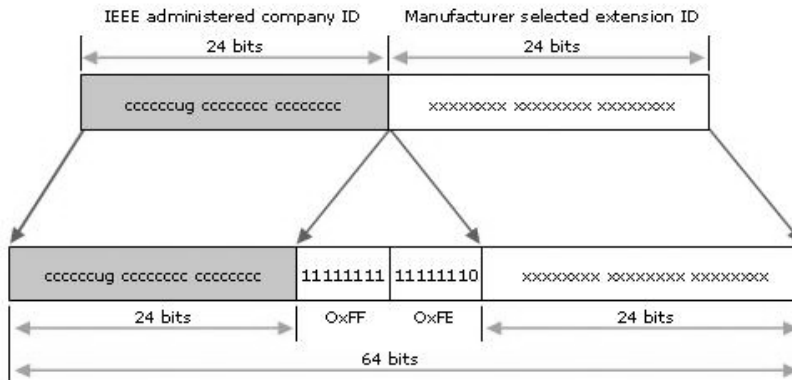
IPv6-Autokonfiguration

Der in IPv6 eingebaute Autokonfigurationsmechanismus ermöglicht eine automatische Adressvergabe, Router-Bekanntmachung sowie Nachbar-Bekanntmachung inklusive Authentifikation, ohne dafür noch wie bei IPv4 eine zentrale Instanz zu benötigen.

Automatische Generierung von Adressen Für die automatische Erzeugung einer link-local Adresse können je nach Layer-2 Medium verschiedene Verfahren angewandt werden.

Ethernet 48-Bit-MAC-Adressen z. B. können auf 64 Bit lange EUI-64 (Extended Unique Identifier) Interface-IDs abgebildet werden:

- die höchstwertigen 24 bit werden auf die höchstwertigen 24 bit der EUI-64-Adresse abgebildet
- die niederwertigen 24 bit werden auf die niederwertigen 24 bit der EUI-64-Adresse abgebildet
- die entstehende 16 bit „Lücke“ wird mit 0xFFFE gefüllt:



- wird nun noch das U/L-Bit (Universal/Local - universell oder lokal verwaltete Adresse) invertiert, erhält man eine 64 bit IPv6-Interface-ID
- zusammen mit einer 64 bit Netz-ID ergibt dies eine gültige link-lokale IP-Adresse

Vermeidung doppelter Adressen Doppelt vergebene Adressen sind bei der *vollautomatischen* Konfiguration (Abilden der MAC-Adresse auf IPv6-Adresse) praktisch unmöglich, allerdings kann es vorkommen, daß einige Knoten im Netz *manuell* zugewiesene IPv6-Adressen besitzen.

Kommt ein Knoten neu ans Netz, kennt er anfangs nur die Multicast-Adresse für alle Knoten am Link-Segment (FF02::1) und seine eigene Multicast-Solicited-Node-Adresse (FF02::1:FFxx:xxxx). Bevor er seine eigene link-lokale IP-Adresse verwenden darf, muss er testen, ob niemand anderes im Netz dieselbe Adresse hat. Der Knoten sendet dazu eine spezielle Neighbor-Solicitation, die im ICMP-Teil die eigene zu überprüfende IP-Adresse enthält:

	Quelle	Ziel
IP	0::0	FF02::1:FFxx:xxxx (eigene Solicited-Node-Multicast-Adresse)
MAC	MAC-Adresse des Sender-Interfaces	33:33:FF:xx:xx:xx (eigene Solicited-Node-Multicast-Adresse)

Die spezielle Absender-IP-Adresse $0::0$ macht diese Nachricht erkennbar als eine Nachricht zur Entdeckung doppelter Adressen.

Trifft nach einer gewissen Zeit keine Antwort ein, kann der Sender annehmen, daß niemand anders im Netz dieselbe IP-Adresse beansprucht. Ist die Adresse jedoch schon vergeben, antwortet der Knoten, der die Adresse hat, mit einem Neighbor-Advertisement. Besagte Adresse wird dann für *beide* Knoten, Fragesteller und Besitzer, obsolet.

Diese Duplikatserkennung wird erstmalig nach einer zufälligen Zeitspanne ausgeführt, um gleichzeitiges Senden von Nachrichten zur Erkennung doppelter Adressen zu vermeiden (z. B. nach einem Stromausfall und gemeinsamer Wiedereinschaltung).

Automatische Generierung von Zufalls-Adressen Da die IP-Adresse eines Interfaces aus seiner weltweit eindeutigen MAC-Adresse gebildet wird, identifiziert damit auch die IP-Adresse eindeutig ein Interface.

Allerdings ist das nicht immer gewünscht, denn so kann der dazugehörige Knoten weltweit eindeutig lokalisiert werden. So könnten bei einem mobilen Gerät (logischer) Aufenthaltsort und Verkehrsmuster ermittelt werden.

Aus diesem Grund existiert eine Erweiterung der automatischen Konfiguration um pseudozufällige IP-Adressen:

Dazu wird aus der 64 Bit-Interface-ID und einem festen (und nur dem Knoten bekannten) 64 Bit-Startwert mittels einer MD5-Hash-Funktion ein 128 Bit Pseudo-Zufallswert erzeugt. Dessen erste 64 Bit werden nun als Interface-ID des Knoten verwendet (Bit 6 als Global/local-Bit auf 0 gesetzt), die weiteren 64 Bit dienen als neuer Startwert für die Generierung weiterer solcher pseudozufälligen Interface-IDs.

Bekanntmachung und Anfrage von Router-Werten Zur vollständigen Konfiguration eines IPv6-Interfaces sind neben der Vergabe einer gültigen IP-Adresse auch Angaben wie Adresspräfix (zur Netzkennzeichnung) und Adressen der vorhandenen Router am Link-Segment nötig.

Dazu machen Router über link-lokale Multicasts periodisch die notwendigen Konfigurationsdaten über so genannte **Router-Advertisement-ICMP**-Nachrichten bekannt. Diese Daten besitzen eine begrenzte Gültigkeitsdauer und müssen vor Ablauf der Gültigkeitszeit erneuert werden. So wird sichergestellt, daß alle Teilnehmer über Veränderungen der Konfiguration auf dem Laufenden bleiben:

	Quelle	Ziel
IP	link-lokale Adresse des Router-Interfaces	FF02:::01 (alle Knoten)
MAC	MAC-Adresse des Router-Interfaces	33:33:00:00:00:01 (alle Knoten)

Kommt ein Knoten kurz nach solch einer periodischen Verteilung der Router-Daten zum Netz hinzu, kann er auch eine direkte Anfrage an die verfügbaren Router stellen, eine so genannte **Router-Solicitation-ICMP**-Nachricht. Der oder die Router antworten dann mit einer oben beschriebenen Router-Advertisement-ICMP-Nachricht.

	Quelle	Ziel
IP	link-lokale Adresse des Sender-Interfaces	FF02::02 (alle Router)
MAC	MAC-Adresse des Sender-Interfaces	33:33:00:00:00:02 (alle Router)

Router Advertisements sind ICMP-Nachrichten und enthalten folgende Informationen, die als Optionen in die ICMP-Nachricht eingefügt werden:

- Gültigkeitsdauer
- Adresspräfix mit Längenangabe (==Subnetzmaske)
- Vorschlag für Hop Limit
- MAC-Adresse des Router-Interfaces
- außerdem kann angegeben werden, ob weitere Konfigurationsdaten von anderen Diensten (z. B. DHCP) geholt werden sollen

Dieses Schema kann auch mit mehreren Routern am selben Link-Segment arbeiten. Knoten im Netz können Nachrichten über einen beliebigen Router verschicken. (Sollte sich der gewählte Router als ungünstig (aufwändigerer Pfad als andere) herausstellen, kann der Knoten über eine ICMP-Redirect-Nachricht instruiert werden, einen anderen Router zu benutzen. Weiterhin entstehen weniger Probleme beim Ausfall eines Routers, das Netz konfiguriert sich (nach Timeout) selbständig um.

Autokonfiguration ohne Router Um mit seinen Nachbarknoten im selben Link-Segment kommunizieren zu können, muss ein Knoten erstens um deren Existenz wissen, zweitens ihre MAC-Adressen kennen.

Die Zuordnung einer Link-Layer-Adresse zu einer IPv6-Adresse erfolgt bei IPv6 *nicht über ARP/RARP*, sondern mittels ICMP.

Dazu gibt es sogenannte **Neighbor-Advertisements**, die entweder unaufgefordert (z. B. beim Hochfahren oder periodisch) oder aufgefordert (als Antwort auf eine Neighbor-Solicitation) versendet werden. Dabei handelt es sich um ICMP-Nachrichten, die als Option die MAC-Adresse des Interfaces enthalten, über das sie abgesendet wurden. Die IPv6-Quelladresse der Nachricht wird dadurch dieser MAC-Adresse zugeordnet.

	Quelle	Ziel
IP	link-lokale IP-Adresse des Sender-Interfaces	FF02::1 (alle Knoten)
MAC	MAC-Adresse des Sender-Interfaces	33:33:00:00:00:01 (alle Knoten)

Eine **Neighbor-Solicitation** dagegen ist eine Anfrage, die benutzt wird, um ein Neighbor-Advertisement zu provozieren. Es soll also zu einer bekannten IP-Adresse die MAC-Adresse herausgefunden werden. Die Anfrage wird mit einem Hop Limit von 255 verschickt, das von bei einem Routerdurchlauf dekrementiert werden würde. Neighbor-Solicitations mit Hop Limit ungleich 255 werden nicht beantwortet:

	Quelle	Ziel
IP	link-lokale IP-Adresse des Sender-Interfaces	FF02::1:FFxx:xxxx (Solicited-Node-Multicast-Adresse des Befragten)
MAC	MAC-Adresse des Sender-Interfaces	33:33:ff:xx:xx:xx (Solicited-Node-Multicast-Adresse des Befragten)

Antworten auf eine Neighbor-Solicitation werden per Unicast nur an den Fragesteller gesendet.

Sicherheit Potenzielle Probleme sind im Zusammenhang mit IP-Spoofing zu sehen. Erlangt ein Eindringling physikalischen Zugang zum Netz, wird er automatisch aufgenommen. So könnten sich beispielsweise böswillige Knoten als Router ausgeben und falsche Konfigurationsdaten verteilen oder auf diese Weise die Kommunikation mit dem echten Router stören. Auch könnten Angreifer auf alle Anfragen zur Duplikatserkennung positiv antworten und so alle Ressourcen, sprich IP-Adressen für sich reservieren.

Die automatische Konfiguration kann deshalb auch mit Authentifikation ablaufen (IP-Sec).

[IPv6-Wiese, IPv6-PrivExt-RFC, IPv6-Interfaces]

DHCPv6

Sollen IP-Adressen statt über das zustandslose dezentrale Verfahren mittels einer zentralen Instanz vergeben werden, kommt wie in IPv4 DHCP zum Einsatz, nun allerdings stark erweitert als DHCPv6. DHCP kann unter anderem zur Vergabe von IP-Adressen verwendet werden und damit die zustandslose Autokonfiguration ersetzen. Außerdem können mit Hilfe von DHCPv6 Konfigurationen vorgenommen werden, die mittels zustandsloser Autokonfiguration nicht möglich sind :

- Verteilung von Rechnernamen
- Zuweisung eines DNS-Servers
- Aktualisierung der DNS-Datenbank bei Namens-/Adresszuteilung
- Verteilung weiterer Netzparameter

Wie DHCP ist auch DHCPv6 ein Client/Server-Verfahren, bei dem ein oder mehrere DHCPv6-Server eine DHCP-Domain verwalten. Der Unterschied zu DHCP besteht darin, daß DHCPv6 gezielt Möglichkeiten von IPv6 (z. B. Multicast- und Anycast-Nachrichten) benutzt, IPv6 ist also als Netzprotokoll für DHCPv6 obligatorisch. Außerdem sind bei DHCPv6 nun die vorher weitgehend herstellerspezifischen „Extensions“ standardisiert worden. Schließlich läuft die Kommunikation nun - anders als bei DHCPv4 - über die UDP-Ports 546 (Client) und 547 (Server).

Funktionsweise Die Konfiguration eines unkonfigurierten Knotens in einem IPv6-Netz kann man in folgendem Schema generalisieren:

1. Lauschen am Link-Segment nach Router-Advertisements (ggf. nach eigener Anfrage).
2. Werden Router-Advertisements empfangen, so können diese auf eine weitere Konfiguration mit DHCPv6 verweisen.
3. Werden keine Router-Advertisements empfangen, muss der Knoten versuchen, sich mittels DHCPv6 zu konfigurieren.

Der betreffende Knoten findet einen DHCPv6-Server nun mittels einer Multicast-Anfrage an alle DHCP-Agenten:

	Quelle	Ziel
IP	link-lokale IP-Adresse des Sender-Interfaces	FF02::1:2 (alle DHCP-Agenten)
MAC	MAC-Adresse des Sender-Interfaces	33:33:00:01:00:02 (alle DHCP-Agenten)

Befindet sich im selben Link-Segment kein DHCP-Server, wird die Suchanfrage von einem DHCP-Relay, daß sich im selben Link-Segment befindet (meist ein Router), an einen DHCP-Server weitergeleitet.

Dieser antwortet mit einem DHCP-Advertisement entweder über ein DHCP-Relay oder direkt an die Unicast-Adresse des Clients. Der wesentliche Zweck dieser Nachricht ist, dem Client die IP-Adresse des Servers zukommen zu lassen. Außerdem kann die Adresspräfixlänge mitgeteilt werden.

Die weitere Kommunikation läuft nach dem schon bei DHCPv4 erwähnten Schema ab.

Sicherheit Im Gegensatz zu DHCPv4 hat DHCPv6 einen Authentifizierungsmechanismus gleich im ursprünglichen Standard vorgesehen. Dieser basiert auf IPv6 und der schon bei DHCPv4 verwendeten „Authentication for DHCP Messages“ (RFC 3118).

[IPv6-Wiese, DHCPv6-RFC]

2.2.2 Namensauflösung

PNRP

Das Peer Name Resolution Protocol ist ein von Microsoft entwickeltes und patentiertes Protokoll für eine dynamische DNS-Namensveröffentlichung und -Auflösung. Es soll damit einen zentralen DNS-Server überflüssig machen und ähnelt daher in seiner Zielrichtung sehr Apples Multicast DNS (mDNS). Leider ist es ein von Microsoft patentiertes, nicht offenes Protokoll. Daher können nur grobe Angaben über die Funktionsweise gemacht werden, detaillierte Informationen fehlen.

PNRP kann sowohl IPv4 als auch IPv6 Adressen auflösen, benötigt jedoch IPv6 als unterliegendes Netzprotokoll. Es ist standardmäßig in Windows Vista integriert.

Mittels PNRP können Knoten in einem IPv6-Netz ihre Namen und dazugehörige IPv6-Adresse veröffentlichen. Darüber hinaus können auch Namen für auf dem entsprechenden Knoten angebotene Services publiziert werden (mittels Portangabe).

Namen können „gesichert“ oder „ungesichert“ veröffentlicht werden. PNRP benutzt dafür asymmetrische Kryptographie.

Funktionsweise

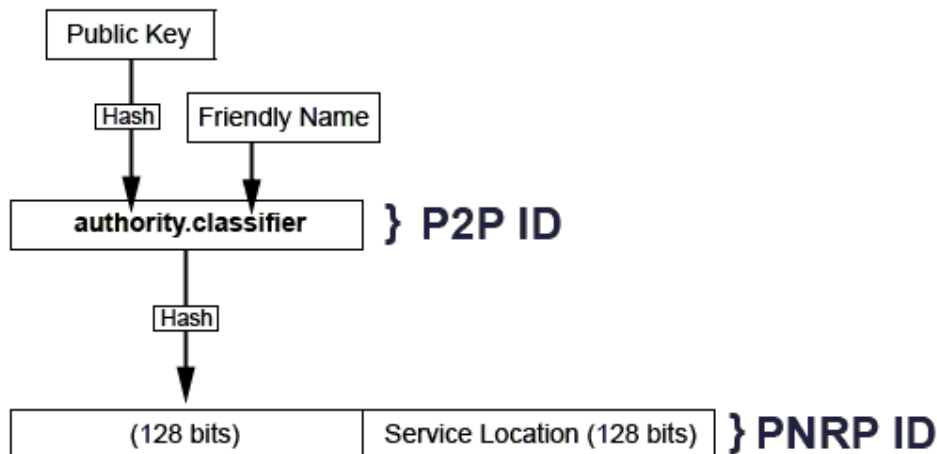
„Clouds“ PNRP definiert bestimmte Gruppen von Knoten, genannt „Clouds“. Solch eine „Wolke“ ist eine Gruppe von Knoten, die sich gegenseitig finden können. Für PNRP existieren dabei die folgenden „Clouds“:

- Eine einzige globale Gruppe mit globalem IPv6 Namensbereich, die alle Knoten im IPv6-Internet repräsentiert.
- Viele link-lokale Gruppen mit link-lokalem Namensbereich, die alle Knoten eines Links repräsentieren, meist das zugeordnete Subnet.

Namen Ein PNRP-Name ist ein Kommunikationsendpunkt und kann einen Knoten, einen Service, einen Benutzer, eine Gruppe oder irgendetwas anderes Bezeichnenswertes mit einer IPv6-Adresse identifizieren.

Diese Namen können gesichert oder ungesichert sein. Unsichere Namen sind normale Strings, ohne irgendwelchen Schutz gegen Spoofing oder doppelte Registrierung. Sichere Namen hingegen sind mit einem Zertifikat und einer digitalen Signatur geschützt.

PNRP-Namen sind 256 Bit lang und folgendermaßen aufgebaut:



- Die oberen 128 Bit sind der eigentliche Name des Endpunktes (P2P ID). Dies ist ein Hash aus „Authority“ und „Classifier“. Für sichere Namen ist „Authority“ der SHA1-Hash des öffentlichen Schlüssels der Namen vergebenden Stelle, für unsichere 0. „Classifier“ identifiziert den Dienst, der adressiert wird, es kann also nach bestimmten Services gesucht werden.
- Die unteren 128 Bit sind eine für jede Instanz einer P2P-ID generierte Zahl.

Für jede „Cloud“ verwaltet nun jeder Knoten einen Cache, der die eigenen und entdeckten PNRP-IDs enthält. Jeder Eintrag in diesem Cache enthält:

- eine PNRP-ID
- eine so genannte „certified peer address“ CPA, ein vom veröffentlichendem Knoten signiertes Datum, daß (IP-)Adressen, Port- und Protokollnummern enthält
- die IPv6-Adresse des veröffentlichenden Knotens

PNRP-Namensresolution ist also das Auflösen von PNRP-IDs zu CPAs.

Namensresolution PNRP benutzt zur Namensresolution eine verteilte Hashtabelle, in der alle Namen einer „Cloud“ verzeichnet sind.

Sie besteht aus zwei Phasen, der Endpunkt-Erkennung und der PNRP-ID-Resolution. Bei der Endpunkt-Erkennung geht es darum, zu einer PNRP-ID die IPv6-Adresse des Knotens ausfindig zu machen, der diese ID publizierte. An diese Adresse kann dann eine PNRP-Anfrage bezüglich der gesuchten PNRP-ID gestellt werden. Die Antwort enthält dann die gesuchte ID und bis zu 4 kB zusätzliche Informationen.

Die Endpunkt-Erkennung basiert auf dem Prinzip verteilter Hashtabellen:

1. Suche im eigenen Cache nach der aufzulösenden PNRP-ID.
2. Ist diese enthalten, ist auch die entsprechende IPv6-Adresse bekannt.
3. Ist die ID nicht im Cache, wird eine Anfrage an den topologisch nächsten (bezüglich des Hashs) Knoten gestellt. Dieser sucht in seinem eigenen Cache nach der ID:
 - a) Wird die ID gefunden, geht sie als Antwort an den Fragesteller zurück.
 - b) Wird die ID nicht gefunden, existiert jedoch im Cache eine ID, die näher ist als die eigene (des Befragten), sendet dieser die dazugehörige IPv6-Adresse an den Fragesteller zurück, der seine Anfrage dann an diesen Knoten stellt .
 - c) Wird die ID nicht gefunden und es gibt im Cache auch keine ID, die näher ist als die eigene, informiert der Befragte den Fragesteller darüber. Der Fragesteller benutzt dann die nächst-nähere ID.

Namenspublikation Vorausgesetzt, der publizierende Knoten hat schon Einträge in seinem Cache, macht er neue PNRP-ID(s) bekannt, indem er Bekanntmachungen an seine (vom Hash her) nächsten Nachbarn sendet *und außerdem* an zufällig aus dem Cache gewählte Nicht-Nachbarn eine Namensresolutions-Anforderung für seine eigene ID stellt. Der Befragte löst daraufhin diese ID zu einer IPv6-Adresse auf und informiert nun wiederum seine eigenen Nachbarn über die neue ID. Auf diese Weise wird eine neue ID auch in (vom Hash her) fernen Bereichen der „Cloud“ publiziert.

Hat der publizierende Knoten jedoch noch keine Einträge im Cache, muss er sich diese entweder:

- aus dem eigenen persistenten Speicher holen (z. B. Festplatte).
- von vom Administrator spezifizierten Verteiler-Knoten holen (z. B. DNS).
- oder mittels Simple Service Discovery Protocol (SSDP) Nachbarn ausfindig machen.

Bewertung PNRP ist also generell ein Protokoll für das effiziente Veröffentlichen und Suchen von Namen (ergo Diensten), allerdings nicht vollkommen selbstkonfigurierend. Für die initiale Nachbarerkennung benötigt es noch fest vorgegebene Einträge oder die Unterstützung anderer Dienste wie DNS oder SSDP.

[PNRP-MSTechnet, PNRP-Wikipedia]

3 Experiment

Verwendete Software

Für die Untersuchung wurden zwei Debian Etch Rechner verwendet, die in Virtuellen Maschinen in *Parallels Desktop for Mac* unter MacOS X 10.4.10 installiert wurden.

Die virtuellen Maschinen wurden auf „Host-Only Networking“ konfiguriert, und können so nur mit dem Host-Betriebssystem und anderen virtuellen Maschinen kommunizieren. Um IPv4LL zu demonstrieren wurde der von Parallels standardmäßig konfigurierte DHCP-Server deaktiviert.

Auf den Debian-Maschinen wurden folgende Pakete installiert:

- avahi-autoipd - für IPv4LL
- avahi-daemon - mDNSResponder der Avahi Zeroconf Implementation
- avahi-dnscconfd - em Namensauflösung über mDNS zu ermöglichen
- avahi-utils - Werkzeuge zum Arbeiten mit dem Avahi Zeroconf Stack

Konfiguration

Die Netzwerk-Interfaces der Debian-Rechner wurden zur Konfiguration mittels DHCP-Konfiguriert, um die Avahi-AutoIP-Konfiguration bei DHCP-timeout zu aktivieren. Die Erforderliche Konfiguration für `/etc/network/interfaces` ist:

```
auto lo eth0
iface lo inet loopback
iface eth0 inet dhcp
```

Um Daten zur Demonstration zu erzeugen, wurden einige Dienste zur Ankündigung mittels DNS-SD konfiguriert. Für jeden Dienst wird eine Datei in `/etc/avahi/services` mit dem Namen `dienstname.service` angelegt.

Auf einem der Rechner wurden folgende Dateien angelegt:

- http.service:

```
<?xml version="1.0" standalone='no'?><!--*~nxml*~-->
<!DOCTYPE service-group SYSTEM "avahi-service.dtd">
<service-group>
  <name replace-wildcards="yes">Webserver on %h</name>
  <service>
    <type>_http._tcp</type>
    <port>80</port>
  </service>
</service-group>
```

- ssh.service:

```
<?xml version="1.0" standalone='no'?><!---nxml-*-->
<!DOCTYPE service-group SYSTEM "avahi-service.dtd">
<service-group>
  <name replace-wildcards="yes">%h</name>
  <service>
    <type>_ssh._tcp</type>
    <port>22</port>
  </service>
</service-group>
```

Auf dem MacOS X Hostcomputer wurde zusätzlich die Netzwerkanalyse-Software „Wireshark2“ installiert, um den Netzwerkverkehr analysieren zu können.

Analyse

IPv4LL Zum Betrachten der Funktionsweise von IPv4LL wird auf dem Hostcomputer Wireshark auf dem Host-Only-Networking Interface gestartet. Der Capture-Filter wird auf arp eingestellt, da IPv4LL lediglich ARP verwendet.

Dann wird der Rechner mit Debian gestartet. Nach dem erfolglosen Versuch eine IPv4 Adresse via DHCP zu beziehen, beobachtet man in Wireshark zunächst einen Ethernet-Broadcast ARP-Request nach 169.254.7.98. Nachdem dieser Request dreimal unbeantwortet bleibt wird sendet der Host einen Gratuitous-ARP für 169.254.7.98 um anzukündigen, daß er diese Adresse nun in Besitz nimmt. Dies wird einmal wiederholt⁶.

mDNS

Aufbau

- Host 1
 - Hostname: foo
 - IP-Adresse: 169.254.7.98
- Host 2
 - Hostname: bar
 - IP-Adresse: 169.254.7.89

⁶Ein Dump der betreffenden Daten ist in der beiliegenden Datei `ipv4ll-analyse.wiresharkcapture` enthalten.

Durchführung Ziel dieses Versuchs ist es, die Anfrage nach dem Rechnernamen `bar.local.` zu dokumentieren.

Dazu wird auf einem der Debian Rechner (mit dem Hostnamen `foo`) das Kommando

```
ping bar.local
```

ausgeführt. Der DNS-Stack auf `foo` versucht daraufhin den Namen `bar.local.` aufzulösen. Dazu sendet er, wie man in Wireshark beobachten kann, eine Anfrage des Typ A nach `bar.local.` an die link-Lokale Multicastadresse (`224.0.0.251`). Host 2 antwortet, ebenfalls an die Link-Lokale Multicastadresse mit dem gefragten Record mit seiner IP-Adresse `169.254.8.89`.

Dann werden die ICMP-Pakete ausgetauscht. Die Implementation des ping-Kommandos fragt daraufhin den Namen zu der Quelladresse der Antwortpakete, damit diese dem Benutzer angezeigt werden kann. Dazu fragt sie nach dem PTR-Record für `89.8.254.169.in-addr.arpa.` und erhält als Antwort einen PTR auf den Namen `bar.local`⁷.

DNS-SD

Aufbau

- Host 1
 - Hostname: `foo`
 - IP-Adresse: `169.254.7.98`
 - Avahi-Konfiguration wie oben beschrieben
- Host 2
 - Hostname: `bar`
 - IP-Adresse: `169.254.7.89`

Ankündigung Auf einem der Debian-Computer wird mittels

```
avahi-publish-service "Testwebserver" _http._tcp 80 "path=/index.html"
```

⁷Ein Dump der betreffenden Daten ist in der beiliegenden Datei `mdns-lookup-und-reverse-lookup-analyse.wiresharkcapture` enthalten.

ein neuer Dienst angekündigt.

Die Analyse zeigt zunächst die Konfliktvermeidung; drei mDNS-Anfragen nach dem Typ ANY und dem Namen `Testwebserver._http._tcp.local..` Diese Anfrage wird dreimal wiederholt und bleibt, wie erwartet, ohne Antwort.

Daraufhin kündigt der Avahi-Daemon diesen Dienst an, indem er, ebenfalls drei mal, eine Antwort versendet, die auf seine eigene IP-Adresse zeigt. Zusätzlich setzt er die Cache-Flush Bits, um die Caches der anderen Link-Lokalen Hosts zu leeren⁸.

Instanzen-Auflistung Mit dem Kommando

```
avahi-browse _http._tcp
```

auf Host 2, der die oben genannten Dienste nicht konfiguriert hat, versuchen wir mittels mDNS und DNS-SD alle link-Lokalen Webserver aufzulisten. Die erwartete Antwort listet die Webserver Instanz auf Host 1 auf.

Dazu sendet Host 2 eine PTR-Anfrage nach `_http._tcp` an die link-Lokale Multicast-adresse mit Zielport 5353 und erhält als Antwort die folgenden Einträge:

- PTR-Record für „Webserver on `foo._http._tcp.local.`“, der anzeigt, daß eine Resource mit dem Namen „Webserver on `foo`“ existiert
- SRV-Record mit dem Port 80 und dem Namen des Dienst-anbietenden Rechners (`foo.local`)
- A-Record, der den Namen `foo.local` in die IP-Adresse von Host 1 auflöst (`169.254.7.98`)

Somit sind alle Informationen zussamen, die ein Nutzen des Webservers auf `foo.local.` ermöglichen⁹.

4 Fazit

Abschließend lässt sich feststellen, dass es eine Vielzahl an gut funktionierenden und spezifizierten Technologien gibt, die konfigurationslose oder konfigurationsarme IP-Netze ermöglichen. Einige dieser Technologien wie DHCP sind omnipräsent, während andere, wie SAP, eher ein Nischendasein fristen.

⁸Ein Dump der betreffenden Daten ist in der beiliegenden Datei `dnssd-ankuendigung-analyse.wiresharkcapture` enthalten.

⁹Ein Dump der betreffenden Daten ist in der beiliegenden Datei `dnssd-dienstauflistung-analyse.wiresharkcapture` enthalten.

Im Bereich der Diensterkennung und Namensauflösung hat sich mit dem Einzug von Multicast und IPv6 viel verändert. Insbesondere die weitere Verbreitung von IPv6 Technologien verspricht, einige Probleme zu lösen.

Leider ist die Vielzahl an Technologien auch eines der größten Probleme. Die meisten Technologien sind zueinander inkompatibel und die Hersteller verzichten aus strategischen Gründen darauf, das Protokoll des jeweiligen Konkurrenten zu unterstützen. So sind sich Microsofts LLMNR und Apples mDNS sehr ähnlich, aber trotzdem nicht interoperabel.

Es bleibt die Hoffnung, dass sich eine Technologie durchsetzen wird, die sich problemfrei in alle Systeme integrieren lässt. Daß dies möglich ist, zeigt der Zeroconf- bzw. Bonjour-Stack von Apple, der als Open Source Software veröffentlicht wurde, mehrere freie Implementierungen¹⁰ besitzt und auf allen großen Plattformen (Windows, MacOS X, Linux und viele BSD-Derivate) komfortabel unterstützt wird.

¹⁰Zu nennen wären hier neben einer Unzahl an Programmierbibliotheken vor allem das Avahi-Projekt, das einen GPL-lizenzierten Zeroconf-Stack entwickelt, der Apples eigener Implementation ebenbürtig ist.

Literatur

- [RARP-RFC] „A Reverse Address Resolution Protocol“, RFC 903: <http://tools.ietf.org/html/rfc903>
- [RARP-Wikipedia] „RARP-Artikel in der Wikipedia“: http://de.wikipedia.org/wiki/Reverse_Address_Resolution_Protocol
- [BOOTP-RFC] „BOOTSTRAP PROTOCOL“, RFC 951: <http://tools.ietf.org/html/rfc951>
- [BOOTP Extensions-RFC] „BOOTP Vendor Information Extensions“, RFC 1497: <http://tools.ietf.org/html/rfc1497>
- [DHCP-RFC] „Dynamic Host Configuration Protocol“, RFC 2131: <http://tools.ietf.org/html/rfc2131>
- [PXE-Draft] „Intel Preboot Execution Environment“ Draft: <http://quimby.gnus.org/internet-drafts/draft-henry-remote-boot-protocol-00.txt>
- [PXE-Spec] „Preboot Execution Environment (PXE) Specification“: <http://www.pix.net/software/pxeboot/archive/pxespec.pdf>
- [IPv4-RFC] „Dynamic Configuration of IPv4 Link-Local Addresses“, RFC 3927: <http://tools.ietf.org/html/rfc3927>
- [mDNS-Draft] „Multicast DNS“ Draft: <http://files.multicastdns.org/draft-cheshire-dnsext-multicastdns.txt>
- [LLMNR-MSTechnet] „Link-Local Multicast Name Resolution“, Microsoft Technet: <http://www.microsoft.com/germany/technet/community/columns/cableguy/cg1106.msp>
- [LLMNR-Draft] „Link-local Multicast Name Resolution (LLMNR)“, Draft: <http://www3.tools.ietf.org/id/draft-ietf-dnsext-mdns-47.txt>
- [LLMNR-RFC] „Link-Local Multicast Name Resolution (LLMNR)“, RFC 4795: <http://www.ietf.org/rfc/rfc4795.txt>
- [DNSSD-AppleDevCon] „Networking - Bonjour“, Apple Developer Connection: <http://developer.apple.com/networking/bonjour/>
- [DNSSD-Draft] „DNS-Based Service Discovery“, Draft: <http://files.dns-sd.org/draft-cheshire-dnsext-dns-sd.txt>

- [DNSSD-LinuxMag] „Null Arbeit - Zeroconf-Netzwerktechniken unter Linux mit Avahi nutzen“, Linux Magazin, Ausgabe 03/2006: http://www.linux-magazin.de/heft_abo/ausgaben/2006/03/null_arbeit
- [SSDP-Draft] „Simple Service Discovery Protocol/1.0“, Draft: <http://quimby.gnus.org/internet-drafts/draft-cai-ssdp-v1-03.txt>
- [SSDP-Wikipedia] „SSDP-Artikel in der Wikipedia“: <http://de.wikipedia.org/wiki/SSDP>
- [SLP-Wikipedia] „SLP-Artikel in der Wikipedia“: http://en.wikipedia.org/wiki/Service_Location_Protocol
- [IPv6-Heise] „Das Mega-Netz“, heise Netze: <http://www.heise.de/netze/artikel/87737/0>
- [IPv6-Wiese] „Das neue Internetprotokoll IPv6“, Herbert Wiese, Hanser 2002
- [IPv6-PrivExt-RFC] „Privacy Extensions for Stateless Address Autoconfiguration in IPv6“, RFC 3041: <http://tools.ietf.org/html/rfc3041>
- [IPv6-Interfaces] „IPv6-Schnittstellenbezeichner“, Microsoft Technet: <http://technet2.microsoft.com/WindowsServer/de/Library/01f5811d-589e-4c11-9161-0ce24a6f8a181031.msp?mfr=true>
- [DHCPv6-RFC] „Dynamic Host Configuration Protocol for IPv6 (DHCPv6)“, RFC 3315: <http://tools.ietf.org/html/rfc3315>
- [PNRP-MSTechnet] „Peer Name Resolution Protocol“, Microsoft Technet: <http://www.microsoft.com/technet/network/p2p/pnprp.msp>
- [PNRP-Wikipedia] „PNRP-Artikel in der Wikipedia“: http://en.wikipedia.org/wiki/Peer_Name_Resolution_Protocol