IPv6 Workshop

Dan Lüdtke

Texte und Graphiken Dan Lüdtke
85386 Eching
Deutschland
Alle Rechte vorbehalten.

Gerätesymbole Lionel Guyot

Veröffentlichung 07. Juni 2017 (3. Auflage)

ISBN-10 1547047380
ISBN-13 978-1547047383

Website www.ipv6-workshop.de

Inhaltsverzeichnis

1 | Einführung

The Internet was designed in 1973 and launched in 1983. In that timeframe, we thought it was an experiment. So we allocated address space – certainly like telephone numbers – sufficient to define 4.3 billion termination points on the Internet. [. . .] It's enough to do an experiment, the problem is the experiment never ended.

Vinton Gray Cerf

1.1 | Über den Workshop

Ziele

Egal ob Sie IPv6 geschäftlich nutzen (sollen) oder sich privat mit dem Thema auseinandersetzen möchten, der IPv6-Workshop wird Ihnen helfen, zügig die notwendigen Grundlagen und die erforderliche Sicherheit in der Anwendung von IPv6 zu erlernen.

Zielgruppe

Dieses Buch richtet sich an alle, die Netze betreuen, vom kleinen Heimnetzwerk bis zum vielschichtigen Unternehmensnetzwerk. Es setzt voraus, dass Ihnen der grundsätzliche Aufbau von IP-Netzen vertraut ist und Sie ein solches regelmäßig nutzen. Sie wissen wo Ethernet im Protokollstapel eingeordnet wird, und die Abkürzungen TCP und UDP sind Ihnen auch ein Begriff. Welche Rolle Nameserver im Internet spielen ist Ihnen bekannt und die Knappheit von Adressen im IPv4-Internet überrascht Sie nicht mehr.

Wenn Sie den oben genannten Aussagen zustimmen können, bringen Sie bereits das erforderliche Grundlagenwissen mit!

Vorgehensweise

Im Workshop wird der Schwerpunkt auf einer verständlichen Beschreibung der technischen Vorgänge liegen. Praxisnahe Beispiele und der Verzicht auf eine vollständige Beschreibung aller Sonderfälle stellen sicher, dass jene Dinge in den Fokus rücken, die für die tägliche Arbeit von Bedeutung sind.

Standardkonformität

Mehr als einmal fällt im Workshop der Begriff *standardkonform*, doch mit den Standards und ihrer Umsetzung ist es so eine Sache. Schon bei anderen Protokollen haben wir erleben können, wie die normative Kraft einer weit verbreiteten Implementierung so manchen Standard korrigierte. Manchmal war dies notwendig um einen stabilen Betrieb zu gewährleisten, in anderen Fällen führte es zu Problemen. Auch bei IPv6 konnten wir ähnliche Tendenzen schon beobachten. So fiel bei der Implementierung von Adress-Timeouts einigen Entwicklern auf, dass ein wort-wörtliches Umsetzen der RFCs in ihrer Kombination in unglücklichen Situationen zu einem undefinierten Zustand führte. Der Vertreter eines namhaften Firewall-Herstellers merkte dazu auf einer Fachkonferenz schnippisch an, es gäbe neben den RFC-Vorgaben auch noch den *richtigen* Weg, IPv6 zu implementieren. All dies verdeutlicht, wie viel Dynamik noch immer in IPv6 steckt. Wenn Sie also beim Ausprobieren der Beispiele ein anderes als das erwartete Verhalten beobachten, sind Sie eventuell Zeuge dieser Dynamik geworden.

1.2 | Über den Autor

Dan Lüdtke ist Ingenieur und Netzwerkexperte mit den Schwerpunkten IT-Sicherheit und IPv6. Während seines Studiums der Elektrotechnik und Technischen Informatik in München gründete er eine IT-Beratungsfirma. Als Betreiber eines eigenen Autonomen Systems konnte er früh Erfahrungen mit IPv6 sammeln, die er in seine Seminare einfließen lässt. In Workshops vermittelt er IPv6 *zum Anfassen*.

1.3 | Systemvoraussetzungen

Im Workshop werden wir viele Features von IPv6 ausprobieren. Einige werden sicher nicht auf Anhieb funktionieren, andere können ein produktives Netz sogar negativ beeinflussen. Es ist also dringend angeraten, den gesamten Workshop getrennt vom produktiven Betrieb stattfinden zu lassen. Durch Virtualisierung von Maschinen und Netzwerken bauen wir uns eine Umgebung auf, in der wir nach Herzenslust experimentieren können.

Hardware

Dafür muss das System, auf dem wir den Workshop durchführen, einige Voraussetzungen erfüllen. Ein moderner Prozessor mit Virtualisierungserweiterung (Intel VT/Vanderpool oder AMD V/Paficia) und 64-Bit-Architektur ist erforderlich. 4 GiB Arbeitsspeicher sind die Mindestanforderung, etwas mehr kann nicht schaden. Auf der Festplatte benötigen wir zwischen 20 und 40 GiB Speicherplatz. Die genaue Menge ist abhängig von der Größe heruntergeladener Dateien und der gewählten Optionen beim installieren der virtuellen Maschinen. Mitwachsende Image-Dateien sparen im Gegensatz zu vorher allozierten Image-Dateien Speicherplatz ein.

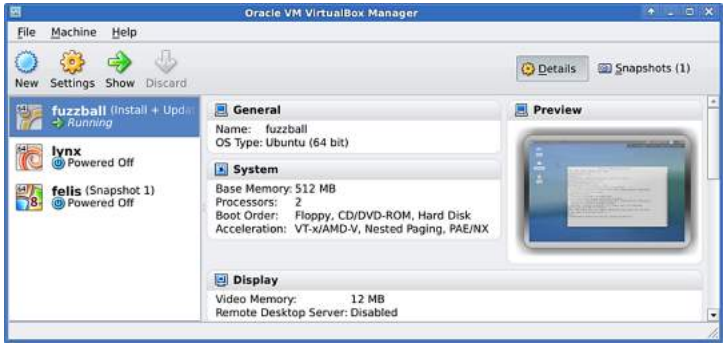
Betriebssystem

Als Betriebssystem kann die 32-Bit- oder 64-Bit-Variante von Microsoft Windows 7 (oder höher) oder ein Linux ab Kernelversion 2.6 dienen. Auch andere Betriebssysteme können genutzt werden, sofern die Desktopvirtualisiserung *VirtualBox* darauf läuft.

Internetzugang

Während des Workshops greifen wir immer wieder auf das Internet zu. Anfangs bei der Installation und dem Update der virtuellen Maschinen und ihrer Programme, später auch um IPv6-Pakete durch das Internet zu tunneln. Dabei reicht es vollkommen aus wenn wir über IPv4 Zugriff auf das Internet haben. Alles was wir an IPv6 brauchen werden wir uns im Workshop mit Hilfe eines Tunnels über die vorhandene IPv4-Infrastruktur in unsere Experimentierumgebung hereinholen.

Abbildung 1.1 Hauptansicht von VirtualBox



1.4 | Software und Werkzeuge

Im Laufe des Workshops werden wir uns ein Lern- und Bastelnetzwerk, bestehend aus verschiedenen Systemen, aufbauen. Mit überwiegend freier Software und unter Einsatz von Virtualisierung, entsteht eine Umgebung, in der gefahrlos experimentiert werden kann. Die eingesetzten Programme und Betriebssysteme werden im Folgenden kurz vorgestellt.

VirtualBox

Um keine unnötigen Kosten entstehen zu lassen, werden wir die kostenlose Desktopvirtualisierung VirtualBox einsetzen. Abbildung 1.1 zeigt die Hauptansicht von VirtualBox. Auf der linken Seite befindet sich eine Liste der verfügbaren virtuellen Maschinen, auf der rechten Seite sind die Parameter der ausgewählten Maschine zu sehen. Die darüber angeordnete Werkzeugleiste enthält Schaltflächen zur Erstellung neuer und Veränderung bestehender Maschinen.

Da ein wenig Übung im Umgang mit VirtualBox nicht schaden kann, sollten Sie sich, sofern noch nicht geschehen, ein paar Minuten mit dem Programm beschäftigen. Erstellen, installieren, modifizieren und löschen Sie ruhig ein paar virtuelle Maschinen um sich mit der Vorgehensweise vertraut zu machen. Alle wichtigen Parameter der im Rahmen des Workshops zu erstellenden virtuellen Maschinen werden an den entsprechenden Stellen genauer beschrieben.

VirtualBox ist als kostenloser Download auf der projekteigenen Website abrufbar.[1]

Wireshark

Ein Blick auf die Leitung verrät oft mehr als jede Dokumentation. Daher werden wir nachschauen wie die gesendeten und empfangenen Pakete tatsächlich aussehen. Mit Wireshark steht uns eine leistungsfähige Software zur Analyse von Netzwerkpaketen zur Verfügung, die plattformübergreifend verfügbar und intuitiv zu bedienen ist.

Wenn wir auf einer Schnittstelle lauschen, werden uns alle Pakete in der Reihenfolge ihres Auftretens im oberen Drittel, der *Paketliste*, angezeigt. Das ausgewählte Paket wird in der darunter angeordneten Ansicht *Paketdetails* und abschließend in der Ansicht *Paketbytes* dargestellt. In den Paketdetails werden die aufgefangenen Daten strukturiert dargestellt. Erkannte Protokolle sind mit Erläuterungen versehen und oft lassen sich die angezeigten Felder ausklappen um weitere Informationen zu erhalten. Markieren wir ein Feld in einem Header, so werden die zugrundeliegenden Bytes in der Ansicht Paketbytes ebenfalls markiert. Wireshark stellt viele Zusatzinformationen bereit, die wir sonst erst mühsam ermitteln müssten. Einen ersten Eindruck von Wiresharks Fähigkeiten vermittelt die Abbildung 1.2.

Falls Sie Wireshark nicht schon von Ihrer täglichen Arbeit her kennen, machen Sie sich ruhig ein wenig mit dem Programm vertraut. Die Dokumentation auf der Projekt-Website ist ausführlich, zahlreiche Videos erleichtern den Einstieg und laden zum Ausprobieren ein. Schneiden Sie doch mal für ein paar Minuten ihren eigenen Internetverkehr mit, vielleicht entdecken Sie bereits IPv6-Pakete.

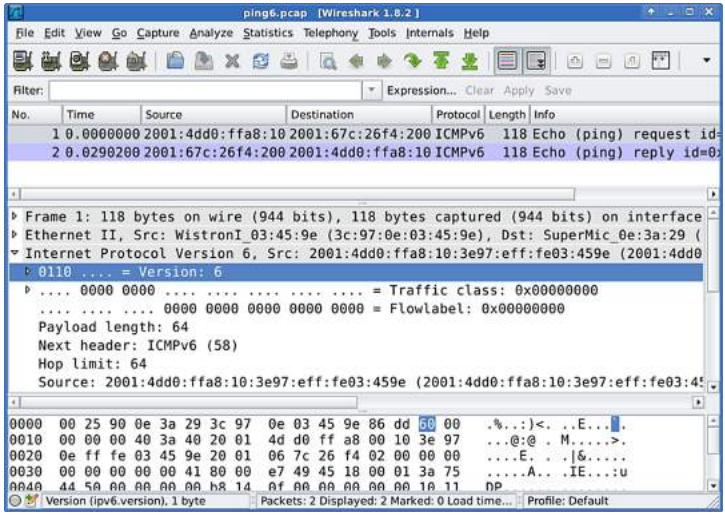
Wireshark kann von der projekteigenen Website kostenlos heruntergeladen werden.[2]

Ubuntu
GNU/Linux
Server

Eine der ersten virtuellen Maschinen die wir erstellen werden, soll später als Router dienen. Die verschiedenen Aufgaben, die ein Router in einem IPv6-Netzwerk übernehmen muss,

[1] *http://www.virtualbox.org*
[2] *http://www.wireshark.org*

Abbildung 1.2 Paketanalyse in Wireshark



lassen sich mit freier Software am einfachsten und verständlichsten erklären. Daher bietet sich das kostenlose Ubuntu GNU/Linux 12.04 LTS der Firma Canonical an.[3] Es basiert auf Debian GNU/Linux, hält aber eine größere, und auch aktuellere, Auswahl an Software-Paketen vor. Die Server-Variante von Ubuntu GNU/Linux ist gegenüber der Desktop-Variante verstärkt auf Sicherheit ausgelegt. Für einen Router, der später im IPv6-Internet präsent sein wird, eine gute Voraussetzung. Auch die Installation dieses Betriebssystems werden wir ausreichend besprechen. Die eingesetzte 64-Bit-Version wird unter anderem auf der Website von Canonical angeboten.[4]

Debian GNU/Linux

Debian GNU/Linux ist nicht nur äußerst stabil, es erfreut sich auch so großer Beliebtheit, dass es kaum eine Frage zu Debian gibt, zu der nicht ein passender Suchmaschinentreffer existiert. Es ist nicht weiter schlimm, wenn Sie mit Debian GNU/Linux noch keine Erfahrungen gemacht haben, ganz nebenbei lernen Sie im Workshop die für Netzwerkadministratoren wichtigsten Schritte kennen. Die Installation werden wir

[3] Die Abkürzung LTS steht für Long Term Support. Dabei handelt es sich um eine Version, die vom Hersteller Canonical noch 5 Jahre nach Erscheinen mit Updates versorgt wird.

[4] *http://releases.ubuntu.com/precise/ubuntu-12.04.2-server-amd64.iso*

ein wenig ausführlicher betrachten. Erfahrene Linux-Anwender können an entsprechender Stelle einfach weiterblättern. Im Workshop werden wir die Netzinstallationsvariante von *Debian GNU/Linux Squeeze* in der 64-Bit-Version verwenden. Diese liegt auf den Servern des Debian-Projektes zum Download bereit.[5]

Microsoft Windows 8

Microsoft Windows konnte schon in der Version 7 relativ gut mit IPv6 umgehen. In Version 8 wurde die Unterstützung noch weiter verbessert. Nach wie vor ist Microsoft Windows ein weit verbreitetes Betriebssystem. Auch wenn sein Betrieb eine kostenpflichtige Lizenz erfordert, und damit nicht kostenlos heruntergeladen werden kann, werden wir einen Blick hinter die Kulissen wagen. Natürlich können Sie diesen Teil auch theoretisch nachvollziehen und so ohne Investition einen Eindruck von Microsoft Windows 8 in IPv6-Umgebungen gewinnen. Im Workshop verwenden wir die englische 64-Bit-Version, die Unterschiede zur deutschen Version halten sich, abgesehen von der Sprache, in Grenzen und sind daher vernachlässigbar.

Auf der Website von Microsoft können Sie mehr über Windows 8 und sein Bedienkonzept erfahren.[6]

Sprache

Sie werden anhand der Beschreibungen und Abbildungen schon bemerkt haben, dass wir im Workshop stets die englischen Versionen der Programme einsetzen. Neben den teilweise holprig wirkenden deutschen Übersetzungen bieten die englischen (internationalen) Versionen weitere Vorteile: Sucht man nach einer englischen Fehlermeldung bei der Suchmaschine seiner Wahl, so erhält man in der Regel mehr aussagekräftige Treffer als mit der Suche nach einer lokalisierten Version derselben Fehlermeldung. Begibt man sich auf die Terminals der eingesetzten Betriebssysteme, landet man zwangsläufig wieder bei der englischen Sprache, in der die Befehle und ihre Parameter verfasst wurden. Wenn Sie lieber eine lokalisierte Variante einsetzen möchten, ist auch das möglich.

[5] *http://cdimage.debian.org/debian-cd/6.0.6/amd64/iso-cd/debian-6.0.6-amd64-netinst.iso*

[6] *http://windows.microsoft.com/de-DE/windows-8/meet*

An einigen Stellen unterscheiden sich die Terminal-Ausgaben dann von denen im Workshop. Die Kommandos sind aber identisch und können ohne Änderung aus dem Workshop übernommen werden.

1.5 | Darstellungen und Lesehilfen

Im Workshop verwenden wir für Paketmitschnitte, Adressen, Maschinennamen, Netznamen, Fachbegriffe, Terminalausgaben und Dateiinhalte unterschiedliche Formatierungen. Dieser Abschnitt dient dazu, diese Elemente voneinander zu unterscheiden und korrekt zu interpretieren.

Paketinhalte

Paketmitschnitte fertigen wir im Workshop mit Wireshark an. Aber nicht immer hat man während des Lesens auch die Möglichkeit eigene Mitschnitte anzufertigen. Alle zum Verständnis wichtigen Pakete sind daher auch als Abbildung vorhanden. Hinweise auf die jeweiligen Abbildungen befinden sich im Fließtext.

Adressen

Adressen, wie zum Beispiel `2001:67c:26f4:800::6:80`, werden mit einer Schriftart fester Breite dargestellt. Dadurch erhöht sich die Lesbarkeit der Adressen, in denen aufgrund hexadezimaler Schreibweise, häufig Buchstaben und Zahlen gemischt auftreten.

Namen und Fachbegriffe

Die Namen von virtuellen Maschinen und von Netzen werden *hervorgehoben* dargestellt. Auch Fachbegriffe werden bei ihrem ersten Auftreten im Text *hervorgehoben*.

Englische Begriffe

Die Informationstechnik ist, wie viele andere Lebensbereiche auch, so durchdrungen von englischen Begriffen, dass eine konsequente Eindeutschung derselben in die Lächerlichkeit führen würde. Unter einer *Freundlichgebetenen-Netzteilnehmer-Gemeinsamen-Adresse* wird sich niemand etwas vorstellen können. Eine *Solicited Node Multicast Address* hingegen findet man in Dokumenten, Programmoberflächen und in Fachgesprächen wieder. Immer wenn es den Lesefluss nicht zu

Datei: /hello/world.txt

```
1 Greetings earthling!
2
3 So long,
4 and thanks for all the fish!
```

Abbildung 1.3 Eine Beispieldatei

sehr beeinträchtigt, werden wie daher die englischen Fachbegriffe aus dem entsprechenden *Request for Comments* (RFC) verwenden.

Terminalausgaben

Wir werden im Workshop überwiegend mit den Terminals der Betriebssysteme arbeiten. Die Ausgaben von Kommandos auf dem Terminal werden wie folgt dargestellt:

```
nutzer@maschine:~$ kommando
Ausgabezeile 1
Ausgabezeile 2
✂ Ausgabezeile 23
Ausgabezeile 24 ist eine ganz besonders lange und nicht ⤸
     enden wollende Zeile, vollgepackt mit Informationen.
Ausgabezeile 25
```

Die Zeilen 3 bis 22 der Beispielausgabe enthalten Informationen die nicht von Bedeutung sind. Diese abzudrucken hätte wertvollen Platz gekostet, darum wurden sie übersprungen. Immer wenn Zeilen übersprungen werden zeigt ein kleines Scherensymbol am Anfang der nächsten Zeile diesen Umstand an. Manchmal sind die Zeilen, die ein Kommando ausgibt, länger als der verfügbare Platz auf der Seite. Solche Zeilen werden umgebrochen und die Stelle des Umbruchs mit einem kleinen Pfeil markiert. Ausgabezeile 24 im obenstehenden Beispiel ist eine solche umgebrochene Zeile.

Konfigurationsdateien

Manchmal ist es notwendig, den Inhalt einer Konfigurationsdatei abzubilden. Die Datei world.txt im Verzeichnis /hello würde, wie in Abbildung 1.3 gezeigt, dargestellt werden.

1.6 | Weiterführende Literatur

Als Ergänzung sei Ihnen, neben dem Studium der aufgeführten Quellen, auch das Buch *IPv6. Grundlagen - Funktionalität - Integration* von Silvia Hagen, erschienen bei Sunny Edition, empfohlen.[7] Es enthält eine nahezu vollständige Beschreibung der in diesem Buch angesprochenen Protokolle und Datenformate.

1.7 | Danksagungen

An der Erstellung dieses Buches waren Personen aus dem engsten Kreise beteiligt. Mit ermutigenden Worten und kritischen Fragen hielten sie das Projekt am Leben. Ihnen, und ganz besonders meiner Familie, sei an dieser Stelle herzlich *Danke!* gesagt.

Besonderer Dank gebührt auch meinen Lektoren Florian Lehner und Jürgen Berrisch. Mit bewundernswerter Akribie lasen sie immer wieder jeden einzelnen Satz. Ihre wertvollen Anmerkungen und kniffligen Fragen beeinflussten den Inhalt des Buches maßgeblich und durchweg positiv.

[7] ISBN-10: 3952294225; ISBN-13: 978-3952294222

2 | Grundlagen

Viele Entscheidungen, welche bei der Entwicklung von IPv6 getroffen wurden, lassen sich im historischen Kontext einfacher nachvollziehen. Daher erfolgt in Abschnitt 2.1 *Die Geschichte einer Revolution* zunächst ein Blick zurück in die Geschichte. Gerade für Leser jüngeren Datums dürfte dies ein spannender Ausflug in die Anfänge global vernetzter Informationstechnik sein. Erfahrene Netzwerker können diesen Abschnitt überspringen. Bei Abschnitt 2.2 *Die Entwicklung von IPv6* sollten aber auch die Nutzer des frühen Internets wieder einsteigen. Die Begriffsdefinitionen in Abschnitt 2.4 schließlich, bilden das Handwerkszeug für das gesamte Workshop und sollten deshalb auf keinen Fall ausgelassen werden.

2.1 | Die Geschichte einer Revolution

Gründung der ARPA

Als die Sowjetunion am 4. Oktober 1957 mit *Sputnik* ihren ersten Satelliten auf eine Erdumlaufbahn brachte, war ihr noch nicht bewusst, dass sie nichts geringeres auslösen würde, als die globale Vernetzung und die damit einhergehende Revolution unseres Kommunikationsverhaltens. Während in Moskau noch fröhlich die Wodkagläser klirrten, erholte man sich auf der gegenüberliegenden Seite des Planeten langsam vom *Sputnik-Schock*. Alsbald entwickelten die US-Amerikaner Strategien um im Rennen der technologischen Vorherrschaft wieder die Führung zu übernehmen. Am 7. Februar 1958 gründeten die Vereinigten Staaten deshalb eine Forschungsagentur

namens *Advanced Research Projects Agency* (ARPA).[1] Neben Überschallfahrzeugen und allerlei gruseligen Anstrengungen, die Leistungen von Soldaten mit Hilfe von Bio-Tuning zu verbessern, brachte die ARPA auch ein Projekt hervor, welches die bis Dato bekannte Welt nachhaltig umsortieren sollte: Das Internet.

Paradigmenwechsel im Vermittlungskonzept

Alles begann mit der Erkenntnis, dass Computer nicht nur als schlichte Rechenmaschinen taugen, sondern sich auch hervorragend für die Kommunikation eignen. Dies geschah just zu einer Zeit, in der das Konzept der leitungsvermittelten Netze bereits bröckelte. Eher widerwillig freundeten sich die damals überwiegend staatlichen Telefongesellschaften mit dem Gedanken an, dass ihre zentral gesteuerten und streng hierarchisch strukturierten Netze den neuen Anforderungen nicht mehr gewachsen sein würden. Wie schwer ihnen dieser Paradigmenwechsel fiel, bewies die Deutsche Bundespost 1980 als sie ihren paketvermittelten Netzzugang namens Datex-P auf eine virtuelle Kanalstruktur fußte. Böse Zungen behaupten gar, mit der nicht enden wollenden Diskussion um Paketpriorisierung und *Quality of Service* (QoS) versuchten die Telekommunikationsunternehmen noch heute ihre alten Geschäftsmodelle zu erhalten.

Geburtstunde des Internets

Schnell erkannte das Militär das Potential eines dezentralen, störunanfälligen Kommunikationsnetzes. Und obwohl die beteiligten Universitäten die Ausgestaltung des Projektes maßgeblich bestimmten, behielten sie stets die Wünsche ihrer Geldgeber aus dem Pentagon im Hinterkopf. Den 29. Oktober 1969 kann man getrost als die Geburtsstunde des Internets bezeichnen. Forscher der *University of California in Los Angeles* (UCLA), der *University of California in Santa Barbara* (UCSB), der *University of Utah* (The U) und des *Stanford Research Institute* (SRI) verbanden die Großrechner ihrer Labore und verschickten mit *LO* die erste Internetnachricht.

[1] Heute spricht man von der *Defense Advanced Research Projects Agency* (DARPA), nach drei Um- und Rückumbenennungen in den 70er und 90er Jahren ist DARPA der aktuelle Name.

Das Internet zeigt seine Stärken

Bereits zwei Jahre später bestand das Internet aus 14 Knoten. Das vorherrschende *Network Control Program* (NCP) regelte als Vorläufer des *Internet Protocol* (IP) den Transport der Pakete zwischen ihnen.[2] Kurz darauf entstanden die ersten Protokolle, die auf NCP als Transportmedium setzten. Die E-Mail-Adressen verloren mit der Einführung des @-Zeichens zur Trennung von Nutzer- und Hostanteil mit der Zeit ihre berühmten *Bang Paths*.[3] Um das Militär bei Laune zu halten entschied man sich 1972 zur Durchführung einer Demonstration. Höhepunkt war die später berühmt gewordenen Konversation zwischen den künstlichen Intelligenzen ELIZA und PARRY, welche über das Internet kommunizierten.[4]

Zuverlässige Datenübertragung

Noch im selben Jahr wurden radio- und satellitengestützte Verbindungen in die alte Welt geschaffen. Selbstverständlich hatten diese ganz eigene Eigenschaften was maximale Paketgröße, durchschnittliche Verlustrate und systembedingte Übertragungskapazität anging. Die Informationsübertragung unter Nutzung der neuen Verbindungen verlor an Verlässlichkeit, dafür musste eine Lösung entwickelt werden. Zwischen 1973 und 1978 entwickelte man in internationaler Kooperation das *Transmission Control Protocol* (TCP) zur zuverlässigen und verbindungsorientierten Übermittlung von Daten in paketvermittelten Netzen. Wie gut TCP funktionierte bewies es 1977 in einem Experiment: Aus einem fahrenden Auto in San Francisco gesendete Pakete wanderten über das ARPANET und mehrere Kabel- und Satellitenverbindungen zur University of Southern California, jedoch nicht ohne einen kurzen Umweg über Norwegen und das Vereinigte Königreich zu nehmen. Im Anschluss an das Experiment wurden TCP und das *Internet Protocol* (IP) voneinander getrennt und sind - zusammen mit dem *User Datagram Protocol* (UDP) -

[2] Die Abkürzung NCP wird oft fälschlicherweise als Network Control *Protocol* interpretiert.

[3] Die Bang Paths dienten der relativen Adressierung von Rechnern. Dabei wurden die Zwischenknoten zum Ziel in der Adresse vermerkt und mit ! (Bang) voneinander getrennt.

[4] Die Unterhaltung ist im RFC 439 [Cer73] dokumentiert.

bis heute als eigenständige Protokolle im IP-Stack anzutreffen.

Institutionalisierung des Internets

Für die inzwischen in DARPA umgetaufte Forschungsagentur war das Experiment ARPANET 1978 beendet und man gab sich mit dem Ergebnis sehr zufrieden. Das Internet indes stand vor seiner nächsten Herausforderung: Der Wandel vom Forschungsnetz hin zum *Netz der Netze*. Kommerzielle wie nichtkommerzielle Teilnehmer, Forschende wie Anwendende, Infrastrukturbetreiber wie Nutzer, sie alle standen in der Verantwortung das Internet weiterzuentwickeln. Das deswegen am *Massachusetts Institute of Technology* (MIT) eingerichtete *Internet Configuration Control Board* (ICCB) wurde im Jahr 1983 - das Internet zählte ca. 500 Hosts - vom *Internet Activities Board* (IAB) abgelöst. Aus dem IAB ging 1986 die *Internet Engineering Task Force* (IETF) hervor, die auch heute noch die Standards für das Internet in sogenannten *Requests for Comments* festhält.

Neuartige Entwicklung von Standards

Mit den RFCs wird ein Entwicklungsprozess gepflegt, der den damals bekannten Modellen entgegen lief. Anstatt in staatlichen oder industriellen Gremien wurden Standards von jedem dem es beliebte in einem offenen Forum, meist in Form von Mailinglisten, diskutiert. Die Mailinglisten standen (und stehen) jedem Interessierten offen. Wer sie abonniert, darf, ungeachtet seiner Herkunft, seines akademischen Grades oder der Finanzkraft seines Arbeitgebers, mitdiskutieren. Dieser Bedingungslosigkeit ist auch der Begriff RFC geschuldet. Denn welcher erfahrene Professor hätte sich schon von Studenten oder jungen Doktoranden den nächsten Internetstandard diktieren lassen? Und so fragte man freundlich nach Kommentaren und Verbesserungsvorschlägen, bis schließlich ein De-facto-Standard entstand. Sobald ein vorgeschlagener Standard ausgereift war und ein gewisser *Konsens* darüber herrschte, wurde er verabschiedet. Ganz ohne formelle Abstimmungen oder Vetorechte staatlicher Organisationen. Damals wie heute bilden die RFCs die Grundlage für die IP-Stacks in Betriebssytemen und allerlei Netzwerkprodukte, vom einfachen Sensor bis zum schrankgroßen Router.

Konkurrierende Modelle

Die *International Organization for Standardization* (ISO) beschloss 1982 mit der Entwicklung eines eigenen Standards zur Verbindung von Netzwerken namens *Open Systems Interconnection* (OSI) zu beginnen. Alsbald gab es auf dem Papier den ersten Entwurf eines Konkurrenten zum bereits funktionierenden TCP/IP-Modell. Als Gremium mit klassischen Arbeitsmethoden war die ISO so behäbig und zugleich ausdauernd, dass sie sich erst in den 1990er Jahren endgültig geschlagen gab. Bis dahin war es üblich in öffentlichen Forschungsprojekten die Anwendung des siebenschichtigen OSI-Modells anstelle des vierschichtigen TCP/IP-Modells zu fordern. Von behördlicher Seite hielt man TCP/IP lange Zeit für eine Übergangslösung, und so wird noch heute das ISO/OSI-Modell gelehrt und gelernt. Nicht, weil man glaubt bald eine umfangreiche Implementierung von OSI zu sehen, sondern vielmehr weil es sich hervorragend dazu eignet, zu verdeutlichen, welche Aufgaben bestimmte Protokolle in ihren jeweiligen Schichten erledigen. Gelegentlich führt dies zu skurril anmutenden Diskussionen, wenn sich zwei Parteien nicht einigen können, in welcher der sieben Schichten ein Protokoll einzuordnen sei. Da es von OSI keine Implementierung mit praktischer Bedeutung gibt, lohnt es sich eigentlich kaum darüber zu streiten.

Stichtag für TCP/IP

Im Forschungsbereich hatten sich zwischenzeitlich verschiedene Netze mit unterschiedlichen Protokollen gebildet, die es nun miteinander zu verbinden galt. Dank guter Vorplanung konnten alle Netze am 1. Januar 1983, dem sogenannten *Flag Day*, auf einen TCP/IP-Stack umgestellt werden. Die vielen Inselnetze sprachen nun ein einheitliches Protokoll, das weitere Wachstum und die Interkonnektivität waren gesichert. Der Siegeszug des Internets konnte beginnen. Zunächst jedoch hauptsächlich außerhalb Europas. Der *Verein zur Förderung eines Deutschen Forschungsnetzes* (DFN-Verein), treibende Kraft der Vernetzung deutscher Hochschulen und Forschungseinrichtungen untereinander, hätte ohne den unermüdlichen Einsatz der Informatik-Rechnerbetriebsgruppe der Universität Dortmund vermutlich noch viele Jahre an OSI festgehalten. Die Dortmunder hatten sich dem TCP/IP-Modell verschrieben,

besaßen eine Verbindung nach Amsterdam und konnten von dort das US-amerikanische Internet erreichen. Damit galten sie als *der* deutsche Anlaufpunkt für alle, die lieber IP nutzten anstatt auf OSI zu warten.

Domain Name System

Dem Wachstum des Internets war ein Problem geschuldet, dass dringend einer Lösung bedurfte: Die Zuordnung von Namen zu IP-Adressen konnte nicht mehr im bis dahin üblichen Verfahren, den Hostfiles, geschehen. Zu häufig und zu umfangreich waren die Änderungen geworden.[5] Die zeitraubende und fehleranfällige Pflege der Hostfiles wurde zugunsten des *Domain Name System* (DNS), ein dezentrales aber hierarchisch aufgebautes Verzeichnis welches den Namensraum des Internets in *Zonen* unterteilt, eingestellt. Die Verwaltung von IP-Adressen wie auch die Koordination der Nameserver der *Root Zone* obliegt seit ihrer Gründung 1988 der *Internet Assigned Numbers Authority* (IANA), einer Abteilung der *Internet Corporation for Assigned Names and Numbers* (ICANN). Heutzutage registriert die IANA auch Protokollnummern, Portnummern und Codes, die in und von Netzwerkprotokollen genutzt werden. Die Ressourcen, hauptsächlich IP-Adressen, werden von der IANA zunächst an eine *Regional Internet Registry* (RIR) und von dort an berechtigte Bedarfsträger, beispielsweise *Internet Service Provider* (ISP), nach regionalen Regeln und Verfahren ausgegeben.[6]

World Wide Web

Mit der Zahl der Hosts wuchs auch die Menge der Inhalte, die im Internet angeboten wurden. Über die verschiedensten Dienste und Protokolle wurden Informationen transportiert, kopiert und verteilt. Was jedoch fehlte war eine einfach zu bedienende, praktische Darstellungsform mit der Möglichkeit, die Inhalte miteinander in Beziehung zu setzen. Bei der *European Organization for Nuclear Research* (CERN) wurde daher *Hypertext* entwickelt, ein früher Vorläufer des *World Wide Web* (WWW). Entsprechende Anzeigeprogramme, die

[5] Tatsächlich besaß jeder Computer eine Datei, in der die Namen anderer Hosts und ihrer IP-Adressen zwecks Namensauflösung hinterlegt waren.

[6] Die RIR für Europa, Russland, den mittleren Osten und Zentralasien ist das *Réseaux IP Européens Network Coordination Centre* (RIPE NCC).

Webbrowser, folgten. Dabei traten sie in Konkurrenz zueinander, und kämpften alsbald im *Browserkrieg* um die Gunst der Anwender und die Definition von Standards. Den vorläufigen Höhepunkt stellt das *Web 2.0* dar, welches sich in technischer Konzeption und Erscheinungsform wahrnehmbar vom ursprünglichen, eher statischen, WWW unterscheidet. Technologien wie Webservices, interaktiv-dynamische Webseiten, die Vernetzung in und mit sozialen Diensten und ein Rollenwechsel des Endanwenders vom reinen Konsumenten hin zu einem produzierenden, teilhabenden Konsumenten, grenzen das Web 2.0 gegenüber seinem Vorgänger ab.

Dotcom-Ära

Mitte der 1990er Jahre hatten alle größeren Akteure aus Politik und Wirtschaft erkannt, dass es ein Internet gibt. Dummerweise folgte dieser Erkenntnis meist Ignoranz, zuweilen gar der Drang, bestimmte Anwendungen oder ganze Technologien verbieten zu lassen. Die Spanne reicht von deutschen Gerichtsurteilen zum Verbot von sexuellen Textbeiträgen über eine Registrierungspflicht für Internetnutzer in China bis hin zu einem später wieder einkassierten US-amerikanischen Gesetz zum Verbot von Schimpfwörtern im Internet. Während das Establishment also damit beschäftigt war, die Kontrolle über etwas Unkontrollierbares zu erlangen, entwickelten kreative Köpfe in Hinterhöfen und Garagen Geschäftsideen, die sich mit dem Fortschritt arrangierten und ihn voran trieben. Es entstanden Unternehmen, die ausschließlich im Internet agierten. Mit wenig Kapital gegründet, waren sie anfangs ein überschaubares Wagnis. Stellten sich jedoch erste Erfolge ein, wurden sie von Investoren schnell mit Risikokapital überschüttet. Viele verloren dabei den Bezug zu betriebswirtschaftlichen Grundlagen und waren faktisch überbewertet. Im Jahr 2000 kam der befürchtete Börsencrash und zahlreiche Start-Ups mussten Insolvenz anmelden. Noch heute wächst das Internet und die darin beheimateten Unternehmen mitsamt ihren Geschäftsmodellen. Nach der schmerzhaften Bauchlandung sind jene, die sie überlebten, jedoch solider aufgestellt und werden jetzt überwiegend nach betriebswirtschaftlichen Grundsätzen geführt.

Die Menschen hinter der Geschichte

Das Internet wurde von vielen klugen Köpfen ersonnen, die in nie dagewesener Art und Weise kooperativ, zielstrebig, kritisch und kritikfähig zusammenarbeiteten. Einige hatten die zündende Idee zur rechten Zeit, andere haben sich kontinuierlich an der Fortentwicklung beteiligt und wieder andere haben mit bewundernswerter Sorgfalt vorgeschlagene Standards gegen unzählige Eventualitäten geprüft. Sie alle haben der Forschung gezeigt, dass man mit Offenheit und Kooperation mehr bewegen kann als mit dem Tüfteln im dunklen Kämmerlein. Sie haben angestaubte Geschäftsmodelle vernichtet und neue erst möglich gemacht, sie haben Menschen aus allen Teilen der Welt miteinander verbunden und unsere Welt nachhaltig verändert. Sie alle beim Namen zu nennen würde den Umfang dieses Werkes sprengen. Erwähnte man nur die vermeintlich Wichtigsten von ihnen, täte man den Weggelassenen Unrecht. Deshalb hat sich der Autor für die namenlose Darstellung der Ereignisse, nicht ohne tiefe Bewunderung für die Akteure, entschieden.

Fazit

Die Geschichte des Internets lehrt uns: Immer dann, wenn eingefahrene Konzepte über Bord geworfen wurden, folgte ein nicht selten unterschätzter technologischer Sprung. Mit IPv6 steht uns der nächste große Sprung ins Haus, und wieder werden lieb gewonnene Verfahren und Denkweisen auf die Probe gestellt. Dieser notwendige, mitunter schmerzhafte, Prozess ist mit einer guten Portion Aufgeschlossenheit zu gut zu bewältigen.

2.2 | Die Entwicklung von IPv6

Irrungen im Versionsdschungel

Schon zwei Jahre vor dem erfolgreichen Umstieg von NCP auf IPv4 erblickte das *Internet Stream Protocol* (ST) das Licht der Welt. Es wurde entwickelt, um die Übertragung von menschlicher Sprache über paketvermittelte Netze zu standardisieren. Um ST- und IP-Pakete voneinander unterscheiden zu können, enthalten ST-Header die Versionsnummer 5, denn praktischerweise fangen sowohl ST- als auch IP-Header mit einem Versionsfeld von 4 Bit Länge an. 1992 wurden die ersten Stim-

men laut, die eine drohende IPv4-Adressknappheit beschworen. Ein Jahr später erschien mit RFC 1475 [Ull93] die Spezifikation von IPv7, TCPv7 und UDPv7. Aufgrund der fälschlichen Annahme, der ST-Nachfolger ST-II belege bereits die Versionsnummer 6, hatte man diese folgerichtig übersprungen. Und so erhielt das deutlich später entwickelte, und erst 1998 in RFC 2460 [DH98] spezifizierte, IPv6 seine inzwischen vom Irrtum befreite Versionsnummer. IPv7 wurde schließlich zugunsten von IPv6 aufgegeben, das nächste IP wird eines Tages vielleicht IPv8 sein, sofern sich nicht wieder jemand im Versionsdschungel verirrt.

Übrigens: Obwohl ST keine ernsthafte Verbreitung fand, beeinflusste es viele heute verwendete Protokolle aus dem Bereich *Voice over IP* (VoIP).

Protokollentwicklung

IPv6 hat bereits eine bewegte Geschichte hinter sich. Es gab zahlreiche Korrekturen und Erweiterungen des Protokolls. Einige wurden zwischenzeitlich als überholt gekennzeichnet und haben nur noch historischen Wert. Circa 15% der Seiten in RFC 2460 [DH98] beschreiben Techniken die inzwischen als veraltet gelten. Das liegt hauptsächlich an Verfahren wie *Loose Source Routing* und *Record Route*, welche Sicherheitsverantwortlichen schon bei IPv4 große Kopfschmerzen bereiteten. Sie wurden wieder abgeschafft und sind heute nur noch bei der Erstellung entsprechender Paketfilterregeln von Bedeutung. In direktem Zusammenhang zu IPv6 steht natürlich das *Internet Control Message Protocol Version 6* (ICMPv6), dem eine essentielle Bedeutung zukommt und das mit RFC 4443 [CDG06] bereits in zweiter Fassung vorliegt. Beide Protokolle zusammen bilden das, was wir heute IPv6 nennen.

IPv6 wird Standard

Ab 2003 nahm die Entwicklung Fahrt auf, zahlreiche Protokolle und Erweiterungen wurden in den Folgejahren in Form von RFCs publiziert. Viele beschäftigen sich mit der Anpassung gewohnter Dienste aus der IPv4-Welt auf IPv6. Es fanden aber auch immer wieder tiefe Eingriffe in das noch junge Protokoll statt. Mit der Zeit kristallisierte sich heraus, welche Dinge sich in der Praxis bewähren würden. Seit 2012 wird von RFC 6540

[GDLH12] als *Best Current Practice* von jedem IP-fähigen Gerät verlangt, IPv6 zu implementieren.

2.3 | Herausforderungen von IPv6

Eine der größten Herausforderungen beim Umgang mit IPv6 ist es, sich eine neue Denkweise anzueignen. Die Erfahrungen, die wir mit IPv4 gemacht haben, bilden dafür ein wertvolles Grundgerüst. Bei der Arbeit mit IPv6 müssen wir aber an einigen Stellen Änderungen an diesem Gerüst vornehmen. Dazu zwei Beispiele:

Ende-zu-Ende-Prinzip

Technologien wie *Source Network Address Translation* (SNAT) und *Port Address Translation* (PAT), wie wir sie aus typischen *Customer Premise Equipment* (CPE) kennen, haben das Ende-zu-Ende-Prinzip des Internets nachhaltig gestört.[7] Die Tatsache, dass ein Host am Internet teilnimmt, dafür aber keine global gültige Internetadresse erhält, hat unser Denken so sehr beeinflusst, dass wir es uns kaum noch anders vorstellen können. Einst aus der Not der Adressknappheit geborene Provisorien sind zum Quasi-Standard geworden. Mit IPv6 ist die Adressknappheit aber kein Thema mehr, uns steht der Weg offen, ein Internet zu gestalten in dem sich Teilnehmer wieder direkt erreichen können. Umwege über entfernte Services oder eine *Cloud* sind zwar weiterhin möglich, aber nicht mehr erforderlich. Stellen Sie sich eine Welt vor, in der zwei Menschen Videotelefonie betreiben können, ohne sich um Port-Weiterleitungen oder Zugangsdaten für einen Videotelefoniedienst kümmern zu müssen.

Ein Host, viele Adressen

In IPv4-Netzen wurde meist angenommen, ein Host sei mit einer IP-Adresse zumindest zeitweise so verknüpft, dass man Rückschlüsse auf ein bestimmtes Gerät oder gar einen bestimmten Nutzer ziehen konnte. Bei IPv6 hingegen ist es durchaus üblich, dass ein *Network Interface Conroller* (NIC)

[7] Mit CPE kann jedes Gerät gemeint sein, dass ein Internetprovider seinen Kunden zwecks Netzzugang bereitstellt. Meist reicht es aus, sich einen Home- oder DSL-Router vorzustellen.

mehrere Adressen hat, und davon hat es sich mindestens eine ohne fremdes zutun selbst zugewiesen. Darüber hinaus hat der Host sich unter Umständen noch Adressen zugewiesen, die eine von ihm selbst bestimmte Gültigkeitsdauer besitzen. Welche Adressen ein Host für wie lange als gültig ansieht, ist von außen kaum zu ermitteln. Wir werden uns spätestens jetzt von der Idee verabschieden müssen, ein Host habe nur eine IP-Adresse. Letzteres war bisher eher der Knappheit und weniger den technischen Grenzen geschuldet.

Neue Herausforderungen?

Was bedeutet dies für die von Ihnen betreuten Netzwerke? Werden Sie nun in der Lage sein, eine schon lange gewünschte Anwendung zu realisieren? Können Sie vielleicht jetzt Altlasten oder unschöne *Workarounds* über Bord werfen? Stellen die erwähnten Veränderungen Ihre Netzwerk- oder Sicherheitskonzepte vor unlösbare Probleme?

Auch wenn es schwerfällt, sollten Sie sich diesen Herausforderungen im Rahmen einer Migration stellen, und sie, wenn möglich, zum Vorteil Ihrer Netze nutzen. Begreifen Sie jede Herausforderung von IPv6 auch als seltene Chance, eine grundlegende Änderung im Netzwerk vorzunehmen.

2.4 | Begriffsdefinitionen

Klar definierte Begriffe sind ein Muss in der Informationstechnik und vermeiden so manches Missverständnis. Darum werden wir einen kurzen Blick auf die wichtigsten Fachausdrücke im Zusammenhang mit IPv6 werfen.

Allgemeine Begriffe

Ein IPv6-fähiges Gerät, sei es ein Handy, ein Computer oder ein Kühlschrank, nennen wir **Node**. Einen Node, der für andere Nodes Pakete weiterleitet, werden wir **Router** nennen. Alle übrigen Nodes bezeichnen wir als **Hosts**.

Als **Link** bezeichnen wir jene Technologien, die in der Lage sind IPv6-Pakete direkt zu transportieren. Dabei handelt es sich meist um Ethernet oder das von Einwahlverbindungen bekannte *Point-to-Point Protocol* (PPP).

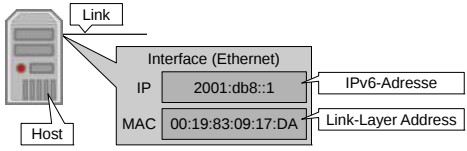


Abbildung 2.1 Interface mit Adressen

Ein Node ist über ein **Interface** mit einem Link verbunden, alle anderen Nodes auf demselben Link sind seine Nachbarn (**Neighbors**).

Zur Kommunikation mit anderen Nodes werden auf einem Link die **Link-layer Addresses** verwendet. Die Länge einer Link-layer Address hängt vom zugehörigen *Link-layer Protocol* ab. Bei Ethernet, zum Beispiel, entsprechen die Link-layer Addresses den MAC-Adressen.

Sofern nicht anders angegeben, meinen wir mit **Adresse** eine IPv6-Adresse. Adressen identifizieren ein am Link angeschlossenes Interface. Ein Interface kann mehrere Adressen haben und es gibt spezielle Adressen, die mehreren Interfaces zugeordnet sind.

In Abbildung 2.1 ist ein Interface mit Link-layer Address und IPv6-Adresse zu sehen.

Protokolle, die von IPv6 in Paketen transportiert werden, ordnen wir der Gruppe Protokolle höherer Schichten (**Upper Layer Protocols**) zu. Sie besteht neben TCP und UDP auch aus ICMPv6.

Wenn wir von einem **Paket** sprechen, meinen wir ein IPv6-Paket, bestehend aus einem IPv6-Header und den Nutzdaten (**Payload**). Sollte ein anderes Paket gemeint sein, so ist dies jeweils angegeben oder ergibt sich zweifellos aus dem Kontext. Eingefleischte Netzwerker müssen sich an dieser Stelle gegebenenfalls umgewöhnen. Ein Paket konnte bisher so ziemlich jede geschlossene Menge an Daten sein. Umgangssprachlich gibt es IP-Pakete genauso wie TCP-Pakete. Nimmt man es aber ganz genau, dann spricht man bei IPv4 von **Datagrammen** (vgl. RFC 891 [Mil83]) und bei IPv6 von Paketen (vgl. RFC 2460 [DH98]). Bei TCP ist die Dateneinheit das **Segment**.

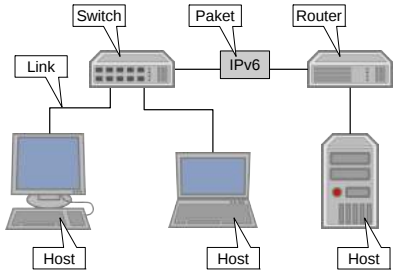


Abbildung 2.2
Begriffe in einem kleinen Netzwerk

Abbildung 2.2 verdeutlicht die Begriffe anhand eines kleinen Beispielnetzwerkes.

Dateneinheiten

Entgegen landläufiger Meinung ist die Definition eines *Bytes* nicht sehr streng, es gibt sogar Prozessorarchitekturen die mit sieben oder neun Bits per Byte arbeiten. Mit dem Oktett hat die ISO deshalb ein genau 8-bittiges Byte beschrieben. Der Einfachheit halber verwenden wir die Begriffe Byte und Oktett im Folgenden synonym und meinen damit stets genau 8 Bits.

Die maximale Größe, die ein Paket annehmen darf, ist von der *Maximum Transmission Unit* (MTU) des transportierenden Links abhängig. Sind zwei Hosts über einen Pfad, bestehend aus einer Folge von Links und Routern, miteinander verbunden, so bildet die MTU des kleinsten Links die **Path MTU**. RFC 2460 [DH98] verlangt, dass die MTU 1280 Bytes nicht unterschreitet, ein IPv6-fähiger Link muss also in der Lage sein Pakete dieser Größe zu transportieren.

Tabelle 2.1 zeigt die in den verschiedenen Schichten bekannter Netzwerkmodelle verwendeten Dateneinheiten.

2.5 | Vorteile von IPv6

Die wichtigsten Features von IPv6 sind im Folgenden kurz erwähnt um einen ersten Überblick zu gewähren. In den jeweiligen Kapiteln werden wir uns intensiver mit den Hintergründen

Tabelle 2.1 Dateneinheiten der verschiedenen Schichten

ISO/OSI-Modell	TCP/IP-Modell	Dateneinheit
Anwendung	Anwendung	Daten
Darstellung		
Sitzung		
Transport	Transport	Segment
Vermittlung	Internet	Paket
Sicherung	Netzzugang	Frame
Bitübertragung		Bit

und den Auswirkungen der einzelnen Eigenschaften beschäftigen.

Adressraum

Die längst überfällige Erweiterung des IP-Adressraums wird mit IPv6 Wirklichkeit. Die von IPv4 gewohnten 32-Bit-Adressen sind damit Geschichte. Mit jeder weiteren Binärstelle verdoppelt sich bekanntermaßen die beschreibbare Menge, das heißt, mit einer Adresslänge von 33 Bits hätte man die Anzahl der IP-Adressen verdoppelt, bei 34 Bits läge bereits die vierfache Menge vor. IPv6-Adressen haben eine Länge von 128 Bits. Wir müssten unser kleines Verdoppelungsbeispiel also 96-mal durchlaufen um diese Zahl zu erreichen. Können Sie sich diese Zahl noch vorstellen? Dem Autor ist es bis heute nicht gelungen! Man darf also annehmen, dass das Thema Adressknappheit für die nächsten Jahrzehnte vom Tisch ist, und genau das war auch die Absicht der Erfinder von IPv6. Kein noch so kleiner, standardkonformer Link soll jemals in die unangenehme Situation geraten, wegen Adressknappheit umgestaltet werden zu müssen. Die Länge der Adressen hat noch einen weiteren, unerwarteten Vorteil: Eine oder mehrere 32-Bit-IPv4-Adressen, Portnummern, Gültigkeitsbereiche oder Flags könnte man theoretisch in den 128 Bit einer IPv6-Adresse unterbringen. Tatsächlich gibt es einige Technologien und Protokolle, die sich darauf verlassen, funktionsrelevante Informationen aus den verwendeten Adressen extrahieren zu können.

Dual Stack

IPv4 wird uns vermutlich noch eine lange Zeit begleiten. Es wäre daher sinnvoll, wenn unsere Geräte sowohl das alte als auch das neue Protokoll *gleichzeitig* verstehen würden. Glück-

licherweise ist das im *Dual Stack*-Betrieb möglich. Anwendungen können explizit eines der Protokolle für eine aufzubauende Verbindung bevorzugen oder die Wahl des günstigsten Protokolls dem Betriebssystem überlassen. Dienste haben sogar die Möglichkeit auf IPv4- und IPv6-Adressen gleichzeitig zu lauschen. Obwohl die beiden Protokolle nicht miteinander kompatibel sind, lassen sie sich bis zur endgültigen Abschaltung von IPv4 bequem gleichzeitig nutzen. Diese Flexibilität erkaufen wir uns freilich mit dem doppelten Administrationsaufwand und der doppelten Pflege von Regeln in Paketfiltern und Netzkomponenten.

Vereinfachter Header

Ein IPv6-Header ist immer 40 Bytes lang. Zwar bieten die sogenannten *Extension Header* die Möglichkeit weitere Informationen anzufügen, doch der wichtigste Teil hat keine variable Länge. Das erlaubt die effiziente Verarbeitung von Headern direkt in Hardware, eine Grundvoraussetzung für den Bau von schnellen Routern. Ungewohnt mag auch die Abwesenheit einer Prüfsumme erscheinen, wir werden später erfahren warum diese nicht mehr benötigt wird.

Stateless Address Autoconfiguration

Abgesehen vom unvorstellbar großen Adressraum ist die *Stateless Address Autoconfiguration* (SLAAC) das wohl bekannteste Feature von IPv6. Dank SLAAC sind Nodes an einem, mit einem Router ausgestatteten, Link in der Lage, sich unter anderem global eindeutige Adressen zuzuweisen. Der Router kann dann Pakete für die Hosts weiterleiten oder ihnen welche aus dem Internet zustellen. Ohne den Einsatz zusätzlicher Protokolle wie *Bootstrap Protocol* (BOOTP) oder *Dynamic Host Configuration Protocol* (DHCP) kann Konnektivität hergestellt werden. Durch die Nutzung von SLAAC können wir uns Teile des administrativen Tagesgeschäfts *wegoptimieren*, allerdings verlieren wir dabei an einigen Stellen auch Protokollierungs- und Kontrollmöglichkeiten.

Privacy Extensions

Bei SLAAC sind ehemals netzlastige Vorgänge wie die Zuteilung einer IP-Adresse in die Logik der Geräte gewandert. Sie folgen aber einem vorhersagbaren Muster, welches Spuren in der Adresse hinterlässt. Dadurch ist es möglich, Rückschlüsse auf die verwendete Hardware zu ziehen und diese

auch über Netzgrenzen hinweg zu identifizieren. Werbetreibende und Bösewichte rieben sich bereits die Hände, doch sie hatten sich zu früh gefreut. Denn die später entwickelten *Privacy Extensions* lösten das Problem auf eine so simple wie effektive Art und Weise. Hosts erwürfeln sich den Host-Anteil einer mit Privacy Extensions erzeugten Adresse zufällig. Nicht nur bei jedem Neustart des Systems, sondern auch in frei konfigurierbaren Intervallen. Ausgehende Verbindungen werden dann bevorzugt mit zufällig generierten Adressen aufgebaut. In der Praxis bedeutet das, dass wir regelmäßig wechselnde Absenderadressen in unseren Netzen sehen werden, die wir nicht ohne weiteres einem bestimmten Host zuordnen können. Das kann die Nachvollziehbarkeit von Vorfällen im Netz erheblich erschweren! Bei Servern hingegen greift man auch bei IPv6 weiterhin bevorzugt zu einer statischen Konfiguration der Adressen.

Multicast

Von kaum zu überschätzender Bedeutung ist in IPv6 das Thema Multicast. Auf Multicast basierende Protokolle erledigen nicht nur die Aufgaben für die sich unter IPv4 das *Address Resolution Protocol* (ARP) zuständig zeigte, sondern sie erlauben eine ganz neue Art und Weise, Nodes am Link anzusprechen. Ein Multicast kann sich auf vordefinierte oder selbstdefinierte Netzbereiche ebenso beziehen wie auf ausgewählte Dienste oder Geräte mit bestimmten Aufgaben. So lässt sich ein Multicast etwa auf alle DNS-Server einer administrativen Einheit oder alle Router an einem Link beschränken. Unzählige weitere, mehr oder weniger sinnvolle, Kombinationen sind möglich. Der gute alte Broadcast, und damit auch die exklusive Broadcast-Adresse, fiel dem zum Opfer: Aus ihm ist ein Multicast, gerichtet an alle Nodes, geworden. Multicast ist zwar nicht grundsätzlich neu, die Gewissheit jedoch, dass alle Nodes am Link diese Technologie beherrschen, war bisher nicht immer gegeben.

Flow Labels

Wie einen Barcode, der einem bestimmten *Datenfluss* zugeteilt wurde, kann man sich die Flow Labels vorstellen. In kontrollierten Umgebungen könnten Flow Labels einem Paketfilter oder einem Router als Entscheidungsgrundlage dienen. Idealerweise sollten alle Pakete mit gleichem Flow Label

auch gleich behandelt werden. Es spricht auch nichts dagegen, echtzeitrelevante Informationen oder QoS-Parameter im Flow Label einzubetten, obgleich für letztere eigentlich ein eigenes Feld im Header existiert. Nach welchen Kriterien ein Flow Label vergeben wird, ist im Einzelfall festzulegen. Die Implementierungen der gängigen Betriebssysteme halten sich dort bislang vornehm zurück. Wer allerdings glaubt, mit den Flow Labels ein leichtgewichtiges *Multi Protocol Label Switching* (MPLS) zu erhalten, wird leider enttäuscht werden.[8] Das Flow Label wird von der Quelle festgelegt und darf auf dem Weg zum Empfänger nicht verändert werden.

IPsec

Hinter dem Begriff *Internet Protocol Security* (IPsec) verbirgt sich eine ganze Sammlung von Protokollen die gesicherte Kommunikation in IP-Netzen ermöglichen. Mithilfe kryptographischer Verfahren und Prüfsummen wird unter anderem der Inhalt vor fremden Blicken und gegen unbemerkte Manipulation geschützt.[9] Der Standard sah ursprünglich vor, dass jedes IPv6-fähige Gerät IPsec implementieren muss. Die Meinungen zur Sinnhaftigkeit dieser Vorgabe gingen auseinander. Die Implementierung von IPsec ist nicht trivial und für das korrekte Funktionieren von IPv6 in vielen Umgebungen nicht notwendig. Eher stiefmütterlich wurde es daher von manchen Entwicklern behandelt. Nicht zuletzt wegen der zusätzlichen Komplexität wurden die Anforderungen später gelockert. Heute sollen Implementierungen nach Möglichkeit IPsec sprechen können, sie müssen es aber nicht mehr zwingend beherrschen.

Multihoming

Ein Link gilt bei IPv6 als *multihomed*, wenn er über mehrere ISP an das Internet angebunden ist. Dabei haben die Interfaces auf einem multihomed-Link Adressen von jedem betei-

[8] MPLS ist eine Technik aus dem Backbone-Bereich. Pakete werden mit einem Label versehen, das den Zielrouter identifiziert. Zwischen den Routern im Backbone wird so ein effizientes Routing realisiert.

[9] Kryptographische Prüfsummen werden auch *Message Authentication Codes* genannt, auf die Verwendung der entsprechenden Abkürzung *MAC* verzichten wir aber um Verwechslungen mit den MAC-Adressen aus *Ethernet* zu vermeiden.

ligten ISP, können also durch die Wahl der Absenderadresse auch den verwendeten Provider selbst bestimmen. Auf diese Weise wird eine Redundanz erzeugt die nicht an bestimmten Routern halt macht, sondern bis in die letzten Ecken des Netzes vordringen kann.

Natürlich ist es auch weiterhin möglich, mit providerunabhängigen Adressen zu arbeiten, und den Adressbereich von mehreren Providern announcen zu lassen.[10] Letzteres stellt ein bewährtes Verfahren zum Multihoming in IPv4-Netzen dar.

Renumbering

Dank *Renumbering* lässt sich ein Link relativ einfach mit neuen Adressen bestücken. Sei es um ein neues Adressschema einzuführen oder um den Wechsel zu einem neuen Provider so *weich* wie möglich zu gestalten.

Dabei ist das Vorgehen am Link denkbar einfach. Ein Link wird übergangsweise in einen multihomed-ähnlichen Zustand gebracht. Nach Migration aller Dienste und (teilweise automatischer) Neukonfiguration der beteiligten Interfaces werden die alten Adressen nicht mehr vergeben. Am Ende besitzen nur noch die neuen Adressen Gültigkeit.

Bei Routern hingegen nutzt man ein spezielles Verfahren namens *Router Renumbering* welches in RFC 2894 [Cra00] spezifiziert wurde. Es erlaubt das Hinzufügen, Ändern und Entfernen von Adressbereichen nach definierbaren Kriterien, was auf Routern mit vielen Interfaces unverzichtbar ist.

Mobile IPv6

IPv6 bringt ausgeklügelte Verfahren zur Umsetzung von *Mobile IP* mit. Dabei handelt es sich um eine Technik, die es erlaubt, mit mobilen Nodes (zum Beispiel Laptops oder Smartphones) unterbrechungsfrei zu kommunizieren, auch wenn diese ihren Aufenthaltsort wechseln. Gemeint ist die Bewegung von einem Link an einen anderen, zum Beispiel vom kabelgebundenen Link in ein Funknetzwerk. Aber auch zwischen Funknetzen ist der Übergang möglich, es entsteht eine Art *Handover* auf der IP-Schicht.

[10] Announcen heißt, sich anderen Netzbetreibern gegenüber für ein Präfix für zuständig zu erklären.

In der einfachsten Variante erhält ein mobiler Node von einem vorher definierten *Home Agent* per Tunnel die an ihn adressierten Pakete zugestellt.

Mobile IPv6 hat sich noch nicht flächendeckend durchgesetzt, wird aber angesichts der wachsenden Zahl mobiler Endgeräte zunehmend interessanter.

Jumbograms

IPv6 *Jumbograms*, nicht zu verwechseln mit den von Ethernet bekannten *Jumbo Frames*, sind Pakete deren Nutzdaten größer als 65.535 Bytes sind. Die Nutzdatenlänge wird eigentlich im Header vermerkt, das entsprechende Feld kann aber keine Werte jenseits von 65.535 annehmen. Mit der *Jumbo Payload Option* im zugehörigen Extension Header sind auch Nutzdaten bis knapp unter 4 Gigabyte möglich. Das setzt allerdings voraus, dass die Path MTU Pakete dieser Größe zulässt. Wir reden hier also eher von Spezialfällen. Im Übrigen müssen Nodes, welche derart gewaltige Links gar nicht bedienen können, zum Beispiel weil sie lediglich Ethernet-Schnittstellen besitzen, diese Option auch nicht unterstützen.

3 | Der Router

Wir werden nun mit dem Aufbau eines kleinen Lern- und Bastelnetzwerkes beginnen. Die erste virtuelle Maschine wird ein Ubuntu Server GNU/Linux sein. Wir werden auf diesem System erste Gehversuche mit IPv6 durchführen und eines Tages soll es als Router fungieren. Widmen wir uns zunächst der Installation der virtuellen Maschine. Dabei haben Sie die Wahl: Sie können die virtuelle Maschine selbst installieren oder als fertige *Appliance* von der Homepage des IPv6-Workshops herunterladen und in VirtualBox importieren.[1] Wenn Sie sich für die Appliance entschieden haben können Sie den folgenden Abschnitt überspringen.

3.1 | Installation des Betriebssystems

Konfiguration der Maschine

Wir starten VirtualBox und erstellen eine neue virtuelle Maschine mit Hilfe des Assistenten, welchen wir aus der Werkzeugleiste heraus aufrufen. Die Maschine wird den Namen **fuzzball** tragen, als Betriebssystem soll **Linux** in der Version **Ubuntu (64 bit)** zum Einsatz kommen. Mit **512 MB** Arbeitsspeicher und einer Festplatte von **8.00 GB** Größe im Format **VDI (VirtualBox Disk Image)** ist der spätere Router gut ausgestattet. Die Festplatte kann platzsparend mit der Option **Dynamically allocated** angelegt werden. Der Speicherplatz auf dem Wirtsystem wird dann erst belegt, wenn das Gastsystem diesen einfordert.

[1] http://www.ipv6-workshop.de/

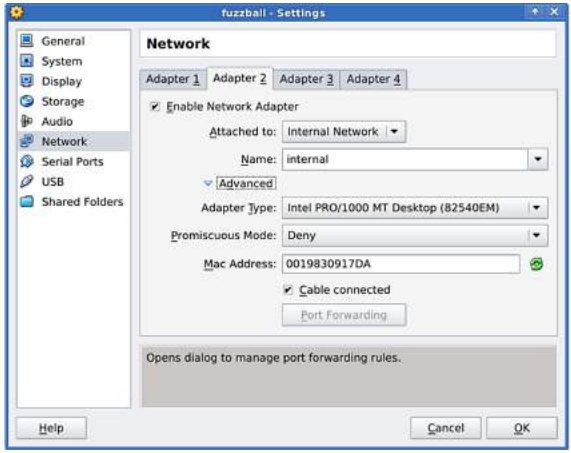


Abbildung 3.1 Konfiguration des zweiten Netzwerkadapters

Die Standardeinstellungen sind in den meisten Kategorien geschickt vorgewählt. Nachbesserungsbedarf gibt es lediglich bei den Netzwerkadaptern. Wir möchten mit zwei Netzwerkadaptern anfangen, daher aktivieren (**Enable Network Adapter**) wir in den Einstellungen der Maschine auch den zweiten Netzwerkadapter. Dieser wird an ein **Internal Network** angeschlossen, dem wir den passenden Namen **internal** verpassen (siehe Abbildung 3.1). Da die MAC-Adresse für einige der späteren Experimente nicht ganz unwichtig ist, verwerfen wir den Vorschlag von VirtualBox und setzen **0019830917DA** als MAC-Adresse ein.

In der Sektion **Audio** deaktivieren wir noch die nicht benötigte Soundkarte indem wir den Haken vor **Enable Audio** entfernen, falls er automatisch gesetzt wurde.

Damit wir mit der Installation des Betriebssystems starten können, legen wir noch das CD-Image von Ubuntu GNU/Linux Server 12.04 LTS in das virtuelle CD-Laufwerk der Maschine ein (siehe Abbildung 3.2).

In Tabelle 3.1 sind die Parameter der virtuellen Maschine zusammengefasst.

Installation des Betriebssystems

Beginnen wir nun die Installation mit der Auswahl von **Install Ubuntu Server** im Menü, das uns nach dem Starten der virtuellen Maschine präsentiert wird. Als erstes werden wir nach

Abbildung 3.2 Einbinden des CD-Images als virtuelles CD-Laufwerk

Virtuelle Maschine *fuzzball*	
Betriebssystem	Linux
Version	Ubuntu Server 12.04 LTS (64 Bit)
Arbeitsspeicher	512 MB
Festplatte	8.00 GB
Netzwerkadapter 1	NAT
Netzwerkadapter 2	Internal Network *internal* MAC-Adresse *0019830917DA*

Tabelle 3.1 Parameter der virtuellen Maschine *fuzzball*

der bevorzugten Sprache gefragt. Im Workshop verwenden wir **English**, Sie können diesen Punkt aber gerne anpassen. Anschließend stellen wir den Ort ein, an dem wir uns befinden, beispielsweise **Other** →**Europe** →**Germany**. Danach fragt uns der Assistent nach der Zeichenkodierung und dem verwendeten Tastaturlayout, wir suchen die passenden Einträge aus der Liste aus und bestätigen sie.

Nach dem automatischen Laden einiger Komponenten verlangt der Assistent wieder nach unserer Aufmerksamkeit. Er möchte das primäre Interface festlegen.[2] Wir bestätigen dazu die Vorauswahl **eth0** und legen als Hostnamen **fuzzball** fest.

Im nächsten Schritt legen wir einen unprivilegierten Nutzer an. Als Nutzernamen verwenden wir im Workshop **user**, es steht

[2] Im Zusammenhang mit VirtualBox sprachen wir von Netzwerkadaptern, auf Betriebssystemebene und in RFCs ist aber von *Interfaces* die Rede. In der Regel repräsentiert ein Interface auch einen vorhandenen Netzwerkadapter. Wir werden von nun an Interface sagen, wenn wir nicht gerade eine virtuelle Maschine in VirtualBox erstellen.

Ihnen selbstverständlich frei einen anderen zu wählen. Das Passwort für den unprivilegierten Nutzer dürfen wir auf keinen Fall vergessen, wir werden es noch häufig brauchen. Eine Verschlüsselung des Home-Verzeichnisses ist nicht erforderlich.

Die Systemuhr konfiguriert der Installationsassistent weitgehend selbstständig. Falls die richtige Zeitzone nicht erkannt wurde, können wir sie jetzt anpassen.

Mit dem vorgeschlagenen Festplattenlayout **Guided - use entire disk and set up LVM** geben wir uns zufrieden. Danach bestätigen wir die zu verwendende Festplatte, den verwendeten Speicherplatz und das finale Schreiben der Partitionstabellen.

Es folgt die Installation der grundlegenden Softwarepakete. Das Installationsmedium enthält jedoch nur die wichtigsten Softwarepakete und ist deshalb darauf angewiesen, die fehlenden Dateien von einem der offiziellen Server nachzuladen. Dazu werden eventuelle Proxy-Informationen abgefragt. Sofern keine besonderen Umstände vorliegen, können wir die Proxy-Informationen frei lassen.

Zwischendurch müssen wir uns für eine Update-Strategie entscheiden. Empfohlen wird die Option **Install security updates automatically**.

Im nächsten Schritt können wir weitere Komponenten selektieren, worauf wir verzichten. Alle benötigten Komponenten und Programme werden wir später manuell nachinstallieren.

Gegebenenfalls fragt der Assistent noch die eine oder andere Information ab, dabei können wir im Zweifelsfall immer bei der Vorauswahl bleiben. Die letzte Frage, die es mit **Yes** zu beantworten gilt, dreht sich um die Installation des Bootloaders. Als letztes lassen wir mit **Continue** das soeben fertig installierte System starten.

Systemupdate

Nach dem Neustart melden wir uns am System als unprivilegierter Nutzer an. Den Nutzernamen haben wir während der

Installation festgelegt, im Beispiel hieß er *user*. Wir wechseln mit dem Kommando `sudo su` zum Nutzer root, dem Nutzer mit den meisten Rechten im System.[3] Softwarepakete werden unter Ubuntu GNU/Linux Server unter anderem mit dem Programm *apt-get* verwaltet. Mit `apt-get` werden wir unser System auf den neuesten Stand bringen. Dazu führen wir erst ein Kommando zur Aktualisierung der Paketinformationen aus:

```
root@fuzzball:~# apt-get update
✂ Reading package lists... Done
```

Anschließend schreiten wir zur Installation der Updates.

```
root@fuzzball:~# apt-get dist-upgrade
Reading package lists... Done
✂ After this operation, 226 MB of additional disk space ⤸
     will be used.
Do you want to continue [Y/n]? y
Get:1 http://de.archive.ubuntu.com/ubuntu/ ⤸
     precise-updates/main linux-image-3.5.0-25-generic ⤸
     amd64 3.5.0-25.39~precise1 [40.4 MB]
```

Dieser Vorgang kann einige Minuten in Anspruch nehmen, und wir müssen der einen oder anderen Aktion gegebenenfalls zustimmen. Das Upgrade schließen wir mit einem Neustart des Systems ab:

```
root@fuzzball:~# reboot
```

Installation der graphischen Oberfläche

Mithilfe von `apt-get` werden wir uns eine graphische Oberfläche installieren. Das ist für einen Router zwar äußerst ungewöhnlich, da wir aber mit graphischen Programmen wie Wireshark arbeiten werden, ist dieser Schritt notwendig:

```
root@fuzzball:~# apt-get install xubuntu-desktop
Reading package lists... Done
✂ After this operation, 1,401 MB of additional disk space ⤸
     will be used.
Do you want to continue [Y/n]? y
✂ Processing triggers for libgdk-pixbuf2.0-0 ...
```

[3] Der Nutzer *root* hat die meisten Rechte auf dem System, er ist der sogenannte *Superuser*.

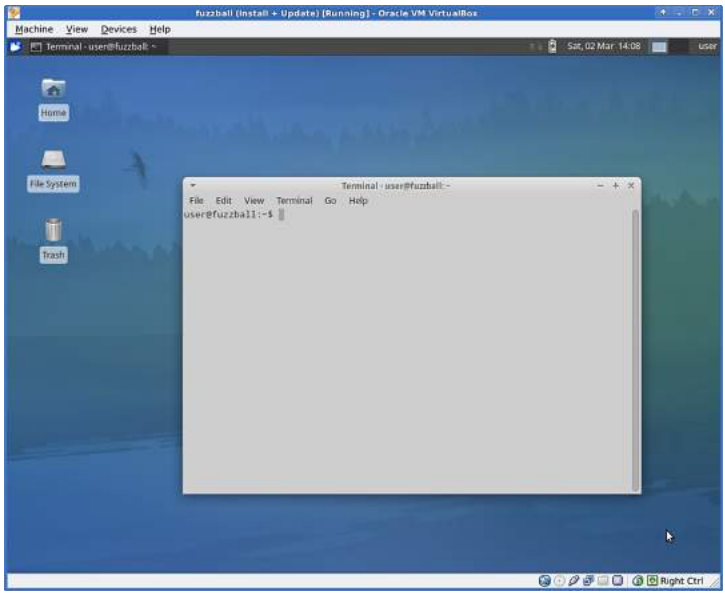


Abbildung 3.3 Desktop mit Terminal nach der Installation

Nach einem Neustart steht uns die graphische Oberfläche zur Verfügung. Wir melden uns an, daraufhin baut sich ein Desktop, wie in Abbildung 3.3 gezeigt, auf.

Wireshark installieren

Im Laufe des Workshops werden wir immer wieder mit Wireshark in Pakete hineinschauen. Deshalb installieren wir das Programm schon jetzt. Dazu öffnen wir ein Terminal und erlangen darin root-Rechte:

```
user@fuzzball:~$ sudo su
[sudo] password for user:
```

Die Installation von Wireshark folgt dem bekannten Schema:

```
root@fuzzball:~# apt-get install wireshark
Reading package lists... Done
After this operation, 62.2 MB of additional disk space ↩
    will be used.
Do you want to continue [Y/n]? y
Setting up wireshark-common (1.6.7-1) ...
Setting up wireshark (1.6.7-1) ...
```

Später werden wir mit Wireshark direkt auf die Interfaces zugreifen, dazu benötigen wir eigentlich root-Rechte. Sicherer

wäre es, das Programm als unprivilegierter Nutzer auszuführen und diesem den Zugriff auf die Interfaces zu erlauben. Das Kommado `setcap` erledigt das für uns:

```
root@fuzzball:~# setcap cap_net_raw,cap_net_admin=eip ↩
    /usr/bin/dumpcap
root@fuzzball:~# getcap /usr/bin/dumpcap
/usr/bin/dumpcap = cap_net_admin,cap_net_raw+eip
```

Network-Manager deaktivieren

Der *NetworkManager* ist ein Programm, welches sich um die Verwaltung von Netzwerkverbindungen und um die Konfiguration der Interfaces kümmert. Dafür nimmt er auch selbstständig Änderungen an den Interfaces vor. Damit könnte er uns beim Arbeiten am Router empfindlich in die Quere kommen. Deshalb stoppen wir ihn:

```
root@fuzzball:~# service network-manager stop
network-manager stop/waiting
```

Damit er auch in Zukunft, zum Beispiel nach einem Neustart, nicht versucht die Kontrolle über die Interfaces an sich zu reißen, müssen wir ihm diese Kompetenz dauerhaft entziehen. Das lässt sich am einfachsten realisieren, in dem wir die Konfigurationsdatei für den automatischen Start von NetworkManager umbenennen:

```
root@fuzzball:~# update-rc.d network-manager disable
update -rc.d: using dependency based boot sequencing
```

Snapshot erstellen

Das System ist nun auf dem neuesten Stand und mit den wichtigsten Werkzeugen ausgestattet. Dies ist ein guter Zeitpunkt um eine Sicherung anzulegen, in VirtualBox auch *Snapshot* genannt. Snapshots dienen der Wiederherstellung eines als funktionierend bekannten Zustandes und sollen uns als Hilfestellung dienen. Dazu fahren wir das System herunter, wechseln auf die Snapshot-Ansicht von VirtualBox, erstellen einen Snapshot mit aussagekräftigen Namen, und fahren das System im Anschluss wieder hoch. Alle nun folgenden Änderungen am System können durch Wiederherstellen des Snapshots rückgängig gemacht werden. Nutzen Sie diese Funktion immer dann, wenn Sie einen Schritt wiederholen möchten.

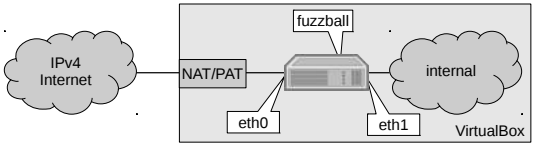


Abbildung 3.4
Links und Interfaces von *fuzzball*

3.2 | Adressen

Interfaces

Die Maschine *fuzzball* ist an zwei Links mit je einem Interface präsent. Abbildung 3.4 zeigt, dass das Interface *eth0* mit dem Internet verbunden ist, das Interface *eth1* ist am Link *internal* angeschlossen. Das Interface *eth0* wird dabei von VirtualBox mit einer IPv4-Adresse aus RFC 1918 [RMK+96] und weiteren verbindungswichtigen Konfigurationsdaten versorgt. Über das Wirtsystem kann das Gastsystem somit auf das Internet zugreifen.

Nun öffnen wir wieder ein Terminal. Zur Manipulation des IP-Stacks und der zugehörigen Interfaces verwenden wir das Kommando `ip`.[4] Die Dokumentation zu `ip` ist mit dem Kommando `man ip` jederzeit abrufbar. Eine lohnende Lektüre! Werfen wir nun mit `ip link show` einen Blick auf unsere Interfaces:

```
user@fuzzball:~$ ip link show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue ⤸
    state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 ⤸
    qdisc pfifo_fast state UP qlen 1000
    link/ether 08:00:27:b7:e4:99 brd ff:ff:ff:ff:ff:ff
3: eth1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state ⤸
    DOWN qlen 1000
    link/ether 00:19:83:09:17:da brd ff:ff:ff:ff:ff:ff
```

Neben den beiden Ethernet-Interfaces taucht ein weiteres Interface auf. Es ist das sogenannte Loopback-Interface *lo* und existiert nur in Software. Dieses Pseudo-Interface dient zum Testen des lokalen IP-Stacks und für den Aufbau von Verbindungen die den Host nicht verlassen sollen.

[4] Eingefleischte Linux-Anwender dürfen auch gerne das etwas ältere `ifconfig` nutzen.

Schauen wir uns dieses Interface ein wenig genauer an, indem wir uns die zugewiesenen Adressen mit `ip addr show dev lo` anzeigen lassen:

Loopback Address

```
user@fuzzball:~$ ip addr show dev lo
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue ⤸
     state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
```

Das Interface *lo* hat zwei IP-Adressen. Eine aus der Klasse *inet* (IPv4) und eine aus der Klasse *inet6* (IPv6). Mit IPv4 kennen wir uns bereits aus. Unser Interesse gilt deshalb der IPv6-Adresse. Sie lautet `::1/128` und sieht wirklich nicht nach dem aus, was man sich unter einer IP-Adresse vorstellen würde. Um diese Schreibweise zu verstehen, folgt ein kleiner Exkurs zur Notation von IPv6-Adressen.

In IPv6 haben die Adressen eine Länge von 128 Bits. Sie sind also eine Kombination aus 128 mal einer Null oder Eins. Das sähe dann zum Beispiel so aus:

Notation von IPv6-Adressen

```
0010 0000 0000 0001 0000 1101 1011 1000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0001
```

Derartige Zahlenkolonnen verlangen nach einer effizienteren Schreibweise. Leider scheidet die dezimale Schreibweise, wie wir sie von IPv4 her kennen, aus Platzgründen aus. Versuchen wir es mit hexadezimaler Notation, kommen wir der Sache schon näher. Je 4 Bits werden zu einer hexadezimalen Ziffer zusammengefasst:[5]

```
2 0 0 1 0 d b 8 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
```

Um die Lesbarkeit zu erhöhen werden dann immer 4 hexadezimale Ziffern zu einem *Block* zusammengefasst. Die

[5] Anhang A *Spickzettel Zahlensysteme* hilft beim Umrechnen der Zahlensysteme.

Blöcke selbst trennen wir mit Doppelpunkten voneinander ab:

```
2001:0db8:0000:0000:0000:0000:0000:0001
```

Diese Schreibweise bietet einige Vorteile. Gerade wenn Adressen viele Nullen enthalten, denn man kann die führenden in den Blöcken bei hexadezimaler Schreibweise bequem weglassen:

```
2001:db8:0:0:0:0:0:1
```

Zusammenhängende Null-Blöcke dürfen maximal einmal mit zwei aufeinander folgenden Doppelpunkten abgekürzt werden:

```
2001:db8::1
```

Diese Adresse lässt sich besser sprechen, schreiben und man könnte sie sich sogar merken.

Uneinheitlichkeit der Notation

Leider ist das Schreiben von Adressen nicht ganz so einfach wie es auf den ersten Blick aussieht, denn RFC 4291 [HD06] erlaubt viele unterschiedliche Schreibweisen. Als Beispiel soll die folgende Adresse dienen:

```
2001:db8:0:0:1:0:0:1
```

Die folgenden Schreibweisen sind gültige Repräsentation der oben genannten Adresse:

- `2001:db8:0:0:1:0:0:1`
- `2001:0db8:0000:000:1:00:0:1`
- `2001:db8::1:0:0:1`
- `2001:db8::0:1:0:0:1`
- `2001:0db8::1:0:0:1`
- `2001:db8:0:0:1::1`
- `2001:db8:0000:0:1::1`
- `2001:DB8:0:0:1::1`
- `2001:0DB8:0:0:1::1`

Hätten Sie sofort erkannt dass es sich immer um die gleiche Adresse handelt? Nun stellen Sie sich einmal vor, sie arbeiten kollaborativ an einer riesigen Liste oder Datenbank. Unterschiedliche Personen tragen dort Adressen ein, jeder nach seinem persönlichen Schreibstil. Sie sollen jetzt feststellen, ob eine bestimme Adresse in der Liste enthalten ist. Nach welcher Schreibweise suchen Sie?

Übrigens: Besonders gemein sind handschriftliche Notizen mit einer Mischung aus Groß- und Kleinbuchstaben. Aus einer `8` wird schnell ein `B` und so manche `0` endete schon als `D`. Gerade bei Adressen die in Paketfilterregeln eingebaut werden sollen, ist das ein sehr unangenehmes Unterfangen.

Verbindliche Notation

Des Problems Lösung versteckt sich in RFC 5952 [KK10]. Dort wird eine einheitliche Notation definiert, die

- in Texten,
- bei der Interaktion von Menschen mit Software,
- zwischen Menschen und
- bei der Darstellung von Adressen durch Software

verwendet werden soll. Nichtsdestotrotz sollen in Eingabefeldern weiterhin alle gültigen Schreibweisen akzeptiert werden.

Notationsregeln

Mit dem Befolgen ein paar einfacher Regeln werden wir den Anforderungen der verbindlichen Notation gerecht. Zur Verdeutlichung folgt jeder Regel jeweils ein Beispiel für die richtige und eines für die falsche Anwendung.

1. Alle führende Nullen innerhalb von Blöcken müssen weggelassen werden:

   ✗ `2001:0db8:00::001`→`2001:db8::001`
   ✓ `2001:0db8:00::001`→`2001:db8::1`

2. Zwei Doppelpunkte müssen immer die größtmögliche Anzahl von Null-Blöcken kürzen.

   ✗ `2001:db8:0:0:0:0:0:1`→`2001:db8::0:1`
   ✓ `2001:db8:0:0:0:0:0:1`→`2001:db8::1`

3. Zwei Doppelpunkte dürfen nie zur Kürzung eines alleinstehenden Null-Blocks verwendet werden.

   ✗ `2001:db8:0:1:1:1:1:1`→`2001:db8::1:1:1:1:1`
   ✓ `2001:db8:0:1:1:1:1:1`→`2001:db8:0:1:1:1:1:1`

4. Sind mehrere gleichwertige Kürzungen möglich, ist die früheste Möglichkeit, von Links beginnend, zu wählen.

   ✗ `2001:db8:0:0:1:0:0:1`→`2001:db8:0:0:1::1`
   ✓ `2001:db8:0:0:1:0:0:1`→`2001:db8::1:0:0:1`

5. Alle alphabetischen Zeichen werden klein geschrieben.

   ✗ `2001:DB8::1`
   ✓ `2001:db8::1`

6. Wenn Portnummern angegeben werden sollen, ist die Adresse in eckige Klammern einzufassen. Der Port muss hinter der schließenden Klammer, mit einem Doppelpunkt abgetrennt, stehen.

   ✗ `2001:db8::1:80`
   ✓ `[2001:db8::1]:80`

Anwendung der Notationsregeln

Wenden wir die eben gelernten Regeln auf die Loopback Address von IPv6 an, bleibt lediglich `::1` übrig.

`0000:0000:0000:0000:0000:0000:0000:0001`→`::1`

Somit ist geklärt warum die Adresse auf Interface *lo* uns so ungewohnt vorkam. Von nun an sind wir in der Lage IPv6-Adressen zu erkennen, gemäß RFC 5952 [KK10] zu notieren und zu kürzen.

Die Unspecified Address

Eine besondere Adresse ist die *Unspecified Address*. Sie besteht ausschließlich aus Nullen und kürzt sich daher auf handliche `::` zusammen. Sie darf nur in wenigen Situationen verwendet werden. Vor allem immer dann, wenn der Absender eines Paketes (noch) nicht im Besitz einer gültigen Adresse ist. Wir werden zu gegebener Zeit auf die Unspecified Address zurückkommen, und merken uns zunächst nur, dass es sie gibt.

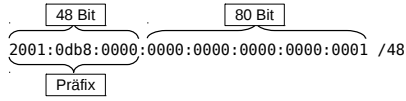


Abbildung 3.5 Adresse mit Präfix

3.3 | Präfixe

Präfixschreibweise

Schauen wir uns nun den Teil hinter der Loopback Address an:

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue ↩
    state UNKNOWN
✂   inet6 ::1/128 scope host
```

Als erstes fällt `/128` ins Auge. Dabei handelt es sich um die sogenannte Präfixlänge, sie wird mit einem Schrägstrich (Slash) von der zugehörigen Adresse getrennt. Präfixe kennen wir schon aus der IPv4-Welt. Dort wurden sie mit dem *Classless Inter-Domain Routing* (CIDR) populär. Anstatt der Netzmaske schreiben wir heute nur noch die Anzahl der Bits hinter die Adresse, die von links beginnend den *Netzwerkteil* der Adresse darstellen. Alle übrigen Bits stehen dann entweder für die weitere Unterteilung in Subnetze oder zur Vergabe an Nodes zur Verfügung, daher heißt der hintere Teil auch *Hostanteil*. Abbildung 3.5 zeigt eine Adresse mit zugehörigem Präfix.

Der vordere Teil der Adresse, auch Netzwerkteil genannt, wird uns in der Regel vom Provider vorgegeben. Der Provider wiederum hat selbst einen Netzwerkteil von einer Registry vorgegeben bekommen. In Europa ist dafür das RIPE NCC zuständig.[6] Die Registry vergibt IPv6-Adressbereiche an jeden der einen berechtigten Bedarf anmeldet und gegebenenfalls nachweist. Natürlich ist der Netzwerkteil, den ein Provider vorgegeben bekommt, wesentlich kleiner als der Netzwerkteil, den er seinem Kunden vorgibt. Die Bits zwischen dem von der Registry erhaltenen Netzwerkteil und dem an den Kunden weitergegebenen Netzwerkteil benötigt der Provider um seine interne Struktur und das Routing abzubilden. Lassen Sie sich bitte nicht verwirren wenn in der Literatur die

[6] http://www.ripe.net

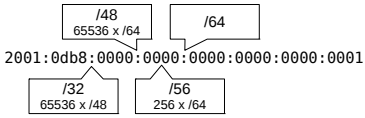


Abbildung 3.6 Typische Präfixlängen

Begriffe Netzwerkgröße und Präfixlänge im selben Satz erwähnt werden. Sie verhalten sich antiproportional zueinander. Je größer das Präfix, desto kleiner das Netz dahinter, und mit absteigender Präfixlänge wächst das dazugehörige Netz.

Präfix am Loopback Interface

Zurück zum Interface *lo*. Dort sind uns also 128 Bits als Präfix vorgegeben, es bleibt also kein Spielraum für Subnetze oder weitere Adressen. Mit dem Präfix /128 ist man daher immer auf eine Adresse beschränkt, nämlich jener Adressen, die vor dem Slash steht. In IPv6 ist nur noch eine Loopback Address vorgesehen. Unter IPv4 stand uns mit `127.0.0.1/8` noch ein beachtlich großes Netz zur Verfügung.

Präfix am typischen Link

Ein typischer Link hat mindestens ein Präfix mit einer Präfixlänge von 64 Bits. Von den 128 Bits, die eine Adresse haben kann, sind dann *nur* 64 Bits dem Präfix zugeordnet. Es bleiben also ebenfalls 64 Bits für Nodes an diesem Link übrig. Rechnen wir kurz nach: 2^{64} Möglichkeiten entsprechen circa $1,84 \times 10^{19}$ (oder 18 Trillionen) Adressen. Das sind nicht nur unvorstellbar viele Adressen, es sind auch mehr als genug für alle nur erdenklichen Situationen, in die so ein Link geraten kann. Das Thema Adressknappheit dürfte erst mal in den Hintergrund treten. Tatsächlich ist ein Präfix von 64 Bits Länge auch das, was man laut RFC 4291 [HD06] an einzelne Links vergeben sollte. Kleinere Links sind nur in besonderen Fällen vorgesehen, und viele der praktischen Features von IPv6 funktionieren mit Präfixen größer oder kleiner als 64 Bits nicht mehr reibungslos. Häufig vorkommende Präfixlängen sind in Abbildung 3.6 dargestellt.

3.4 | Address Scopes

Das Kommando `ip` zeigt uns noch eine weitere Information für die Loopback Address an: Host Scope

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue ⤸
     state UNKNOWN
✂    inet6 ::1/128 scope host
```

Hinter dem Präfix stehen die Worte *scope host*. Abgesehen von der Unspecified Address, hat jede Adresse einen sogenannten *Scope* oder *Gültigkeitsbereich*. Der Scope ist der Teil eines Netzwerkes in dem die zugehörige Adresse als gültig anerkannt wird. Ein Scope identifiziert also einen Bereich eines Netzwerks. Offensichtlich wird die Adresse `::1` außerhalb unseres Hosts nicht als eine gültige Adresse zur Adressierung des Interfaces *lo* anerkannt. Kaum verwunderlich, hat doch jeder Node ein eigenes Loopback Interface welches ebenfalls die Adresse `::1` nutzt. Mit dem Loopback Interface können wir auch keine Pakete an andere Nodes versenden, schließlich liegt dem Interface nicht einmal ein physischer Netzwerkadapter zugrunde. Hosts, die auf einem regulären Interface ein Paket mit der Loopback Address als Ziel erhalten, müssen das Paket verwerfen. Router dürfen Pakete dieser Art niemals weiterleiten.

Die beiden wichtigsten Scopes sind *link-local* und *global*. Der oben genannte Scope *host* gehört genaugenommen zum Scope *link-local*, das Kommando `ip` erkennt jedoch dass dem Interface kein physischer Netzwerkadapter zugrunde liegt und benennt den Scope eigenmächtig um. Weitere Address Scopes

Nicht zu verwechseln sind die Adress Scopes mit den Multicast Scopes mit denen wir uns in Abschnitt 4.5 *Multicast* beschäftigen werden.

3.5 | Link-local Addresses

Interface hochfahren

Wenden wir uns nun dem Interface *eth1* zu. Es befindet sich noch im Zustand *DOWN*, wir müssen es erst hochfahren:

```
root@fuzzball:~# ip link set up dev eth1
```

Wir geben dem Interface ein wenig Zeit hochzufahren und nach circa 10 Sekunden schauen wir uns die Adresskonfiguration des Interfaces mit `ip addr show dev eth1` an:

```
user@fuzzball:~$ ip addr show dev eth1
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 ⤸
     qdisc pfifo_fast state UP qlen 1000
    link/ether 00:19:83:09:17:da brd ff:ff:ff:ff:ff:ff
    inet6 fe80::219:83ff:fe09:17da/64 scope link
       valid_lft forever preferred_lft forever
```

Das Interface hat bereits eine Adresse, sie lautet `fe80::219:83ff:fe09:17da` und hat eine Präfixlänge von 64 Bits. Die Adresse liegt damit innerhalb des Link-local-Präfixes `fe80::/10`. Da wir uns nicht erinnern können dem Interface diese Adresse zugewiesen zu haben, muss sie automatisch generiert worden sein.

Generierte Adressen

Jedes Interface, das an einen Link angeschlossen wird, generiert sich selbst eine Link-local Address und verwendet diese für die Kommunikation mit anderen Nodes auf demselben Link. Das heißt, ohne manuelle Konfiguration oder Zutun von Diensten wie DHCP, haben alle Nodes an einem Link die Möglichkeit über IPv6 miteinander zu kommunizieren. Gerade in Ad-Hoc-Netzen über *Wireless LAN* (WLAN) ist diese Funktion ein echter Segen: *Schnell mal zwei Rechner miteinander verbinden* war nie einfacher.

Link-local Addresses und Interfaces

Wir werden die Erreichbarkeit unserer Link-local Address mit einem Echo Request auf die Probe stellen:

```
user@fuzzball:~$ ping6 -c 3 fe80::219:83ff:fe09:17da
connect: Invalid argument
```

Wie konnte das schiefgehen? Wir haben eben gelernt, dass alle Link-local Addresses aus dem Präfix `fe80::/10` stammen und dass jedes Interface eine solche Adresse haben muss. Darüber hinaus kann ein Node mehrere Interfaces besitzen, gerade Router verfügen häufig über eine beträchtliche Anzahl an Interfaces. Es ist zwar unwahrscheinlich, aber eben möglich, dass es an mehreren Links einen Node mit derselben Adresse gibt. Solange wir dem System nicht mitteilen über welches Interface eine Link-local Address erreicht werden soll, kann das System keine Pakete an diese richten. Wir müssen `ping6` also neben der Adresse auch noch das Interface mitgeben, über das es sich selbst Echo Requests senden soll. Dazu schreibt RFC 4007 [DHJ+05] vor, dass die beiden Informationen mit einem %-Zeichen voneinander getrennt werden sollen.[7] Das neue Ziel für `ping6` lautet also `fe80::219:83ff:fe09:17da%eth1`:

```
user@fuzzball:~$ ping6 -c 3 fe80::219:83ff:fe09:17da%eth1
PING fe80::219:83ff:fe09:17da%eth1 ⤸
     (fe80::219:83ff:fe09:17da) 56 data bytes
64 bytes from fe80::219:83ff:fe09:17da: icmp_seq=1 ⤸
     ttl=64 time=0.081 ms
✂ 3 packets transmitted, 3 received, 0% packet loss, time ⤸
     1998ms
```

Nun klappt es auch mit den Echo Requests. Übrigens, die ungewohnte Schreibweise mit %-Zeichen und Interface brauchen wir in der Praxis eher selten. Die meisten Ziele, mit denen wir kommunizieren, sind durch eine Adresse außerhalb des Link-local Scopes eindeutig beschrieben. Die Entscheidung, über welches Interface ein Paket unseren Node verlässt, wird im Regelfall also weiterhin vom Betriebssystem getroffen.

Permanente Konfiguration

Damit das eben konfigurierte Interface auch nach einem Neustart automatisch hochfährt und wir nicht jedes Mal manuell eingreifen müssen teilen wir dem Betriebssystem mit, wie das

[7] In RFC 4007 [DHJ+05] wird bevorzugt das Wort *Zone* für Bereiche mit überlappenden oder gleichen Präfixen verwendet. Da eine Zonengrenze aber stets an einem Interface enden muss, hat es sich eingebürgert, das Interface als Alias für die Zone herzunehmen.

Abbildung 3.7 Permanente Konfiguration von *eth1*

Datei: `/etc/network/interfaces`

```
1  # Loopback Interface
2  auto lo
3  iface lo inet loopback
4
5  # Interface an VirtualBox auf dem Wirtsystem
6  allow-hotplug eth0
7  iface eth0 inet dhcp
8
9  # Interface am Link "internal"
10 auto eth1
11 iface eth1 inet6 manual
12         up ip link set up dev eth1
13         down ip link set down dev eth1
```

Interface zu konfigurieren ist. Dies geschieht bei Ubuntu GNU/-Linux in der Datei `/etc/network/interfaces`. Wir editieren die Datei als Nutzer root im Editor unserer Wahl, zum Beispiel mit:

```
root@fuzzball:~# nano /etc/network/interfaces
```

Im Anschluß sollte die Datei, Kommentare ausgenommen, wie in Abbildung 3.7 dargestellt aussehen.

3.6 | Ein Tunnel ins IPv6-Internet

Was wir bisher noch nicht gesehen haben, sind Adressen des Scopes *global*, denn es fehlt uns noch der Zugang zum IPv6-Internet. Das werden wir jetzt nachholen.

Sollten Sie in der glücklichen Lage sein, bereits von Ihrem Internetanbieter mit einem ausreichend großen Präfix ausgestattet worden zu sein, dann können Sie sich die Einrichtung des Tunnels sparen.[8] Wie genau Sie das Präfix bis zum Interface *eth0* der Maschine *fuzzball* routen, hängt stark von Ihrer individuellen Konfiguration und vom verwendeten Internetanbieter ab.

[8] Mehrere /64-Präfixe sollten es mindestens sein, besser ist ein /56-Präfix. Die Angaben in den folgenden Kapiteln sind dann jeweils an die Gegebenheiten anzupassen.

Im Rahmen des Workshops werden wir uns daher mittels eines Tunnels Zugang zum IPv6-Internet verschaffen und dessen Einrichtung ausführlich diskutieren.

Testen der Konnektivität

Im Moment gelingt *fuzzball* dank des in VirtualBox eingebauten NAT der Zugriff auf das IPv4-Internet, vorausgesetzt natürlich, das Wirtsystem selbst ist online. Ein beliebtes Vorgehen, um die Konnektivität eines System zu testen, ist das Versenden von Echo Requests an die öffentlichen Nameserver von Google:

```
user@fuzzball:~$ ping -c 3 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_req=1 ttl=63 time=266 ms
✂ 3 packets transmitted, 3 received, 0% packet loss, time ↩
    2643ms
```

Überprüfen Sie die Einstellungen des Gastsystems, seiner Netzwerkadapter und die Internetverbindung des Wirtsystems wenn dieser Test wiederholt und selbst mit anderen Zielen fehlschlägt! Eine funktionierende Internetverbindung ist Voraussetzung für das Funktionieren der folgenden Schritte.

Tunnelbroker

Eine Möglichkeit IPv6-Pakete zu versenden und zu empfangen, obwohl keine native IPv6-Konnektivität besteht, ist Tunneling. Dabei werden IPv6-Pakete als Payload in IPv4-Datagramme verpackt und dann über die funktionierende IPv4-Infrastruktur zu einem Tunnelserver transportiert. Dort werden die IPv6-Pakete wieder aus den IPv4-Datagrammen herausgelöst und gelangen über die Anbindung des Tunnelservers in das IPv6-Internet. Ein Antwortpaket geht den entgegengesetzten Weg und landet so schließlich beim ursprünglichen Absender des ersten Paketes. Neben funktionierender IPv4-Anbindung setzt dieses Vorgehen auch das Vorhandensein eines Tunnelservers, betrieben von einem sogenannten *Tunnelbroker*, voraus.

Durch die Nutzung eines Tunnelbrokers können wir uns über die bestehende IPv4-Infrastruktur einen kleinen Teil des IPv6-Internets auf *fuzzball* holen. Das Prinzip ist in Abbildung 3.8

Abbildung 3.8 Nutzung eines Tunnelbrokers und VirtualBox

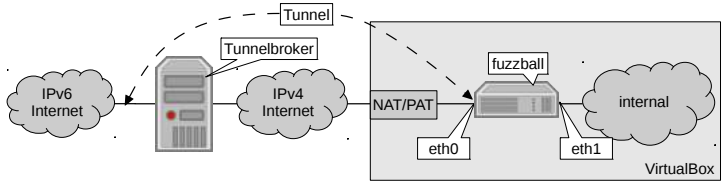


Tabelle 3.2 Bekannte Tunnelbroker

Anbieter	Website	6in4	AYIYA
SixXS	*www.sixxs.net*	✓	✓
Hurricane Electric	*www.tunnelbroker.net*	✓	✗
gogo6	*www.gogo6.com*	✓	✗

dargestellt. Die drei bekanntesten Tunnelbroker werden in Tabelle 3.2 aufgelistet.

Viele Tunnelbroker nutzen das in RFC 3053 [DFGL01] spezifizierte Verfahren welches Protokoll 41 (auch bekannt als *6in4*) verwendet. Es hat den Nachteil, nicht durch NAT/PAT-Router hindurch zu funktionieren, und ist damit für unseren Aufbau nicht geeignet. Wir benötigen daher einen Anbieter der das Protokoll *Anything In Anything* (AYIYA) verwendet, wie zum Beispiel das SixXS-Projekt.

Projekt SixXS

SixXS ist ein Tunnelbroker der Endnutzern kostenlose Tunnel zur Verfügung stellt. Das Projekt ist nicht gewinnorientiert und wird von namhaften Sponsoren getragen. Ziel ist der einfache Zugang zu IPv6 für Netzwerkspezialisten, Entwickler und alle anderen Interessierten. Der Name steht für Six Access. Frei übersetzt also den Zugang zu IPv6.

Neben einem Tunnel erhalten registrierte Nutzer dort auch ein /64-Präfix, welches sich durch den Tunnel routen lässt. Weitere Tunnel oder größere Präfixe lassen sich mit einer virtuellen Währung namens *IP SixXS Kredit* (ISK) erstehen. Mit jeder Woche, die ein Tunnel online ist, wird dem Nutzerkonto ein bestimmter Betrag in ISK gutgeschrieben. Es ist weder nötig, noch möglich, ISK zu kaufen. Das heißt, das gesamte Angebot kann kostenlos genutzt werden.

Registrierung bei SixXS

Der Betrieb eines Tunnels verspricht unkomplizierten Zugang zum IPv6-Internet. Und das unabhängig vom Stand der IPv6-Konnektivität des eigenen Internetanbieters. Gleichzeitig bedeutet der Betrieb eines Tunnels aber auch eine gewisse Verantwortung. Um den leider immer wieder vorkommenden Missbrauch der kostenlosen Angebote einzudämmen, verlangt SixXS recht umfangreiche Angaben bei der Registrierung. Sind die Angaben fehlerhaft oder falsch, erfolgt keine Freischaltung. Daher sind wahrheitsgemäße Angaben bei der Registrierung Pflicht. Wer sich um seine Privatsphäre sorgt, kann einige der Angaben durch SixXS vor der Öffentlichkeit verstecken lassen.

Unter der Adresse *https://www.sixxs.net/signup/create/* registrieren wir uns. Als Begründung geben wir zum Beispiel ein: *I would like to register in order to follow the instructions from the IPv6-Workshop, which requires a SixXS tunnel.*

Nun müssen wir auf die Freischaltung durch die Administratoren von SixXS warten. Da SixXS ehrenamtlich betrieben wird, und die Administratoren oftmals nicht sofort reagieren können, sollten wir ein wenig Geduld mitbringen. Meist erfolgt die Freischaltung innerhalb weniger Stunden.

Einen Tunnel beantragen

Nach der Freischaltung melden wir uns auf der Homepage von SixXS an und begeben uns in den Bereich *Home*. Von dort aus können wir unsere Tunnel (sobald wir welche haben) und die zugehörigen Präfixe (Subnetze) verwalten. Der Menüpunkt *Request Tunnel* bringt uns zum Formular für die Beantragung eines Tunnels. Der erste Schritt ist in Abbildung 3.9 zu sehen. Hier ist es wichtig, neben dem Ort auch die Option **Dynamic NAT-traversing IPv4 Endpoint using AYIYA** anzugeben.

Im nächsten Schritt suchen wir uns einen nahegelegenen *Point of Presence* (PoP) aus, idealerweise einen vom eigenen Internetanbieter betriebenen (vergleiche Abbildung 3.10). Als Begründung für den Antrag kann der oben genannte Text, in leicht abgewandelter Form, dienen: *I would like to request*

Abbildung 3.9 Beantragen eines Tunnels: Schritt 1

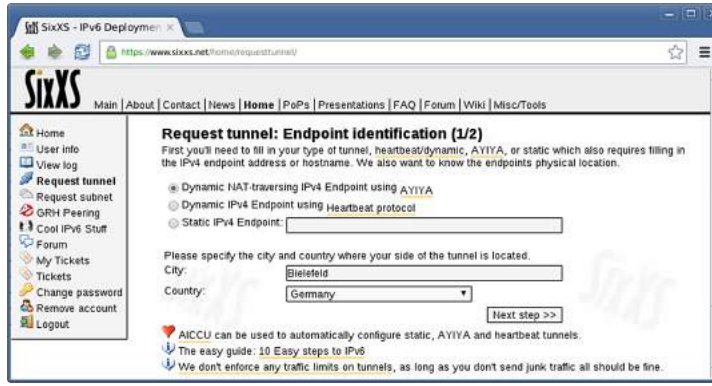


Abbildung 3.10 Beantragen eines Tunnels: Schritt 2

a tunnel in order to follow the instructions from the IPv6-Workshop.

Nun heißt es wieder warten. Wie die Registrierung erfordert auch die Freischaltung des Tunnels das Mitwirken eines Administrators. Es spricht allerdings nichts dagegen, in der Zwischenzeit in den anderen Abschnitten dieses Buches zu stöbern.

Sobald die Freischaltung erfolgt ist, wird der Tunnel im *Home*-Bereich von SixXS sichtbar, vergleichbar mit der Darstellung in Abbildung 3.11. Die angesparten ISK werden zu Anfang noch einen niedrigen Wert aufweisen. Dieser steigt aber mit jeder Woche die der Tunnel aufrechterhalten wird. Wenn ein Tunnel aus der virtuellen Umgebung in einen Home Router oder ein

Abbildung 3.11 Tunnel- und Subnetzübersicht

anderes CPE wandert, steigt die Zahl der ISK auf dem Konto praktisch im Schlafe.

Bitte nehmen Sie sich etwas Zeit um sich mit der Oberfläche von SixXS vertraut zu machen. Zwar sind alle essentiellen Schritte im Workshop ausreichend bebildert, schaden kann ein generelles Verständnis der vielen Daten und Optionen, die ein Tunnel mit sich bringt, trotzdem nicht.

Einrichten des Tunnels

Auf jeder Seite eines Tunnels gibt es ein Tunnel-Interface, so auch bei den Tunneln von SixXS. Wieder einmal repräsentiert ein Interface hier nicht einen physischen Netzwerkadapter, sondern einen virtuellen. Die Software, welche das Tunnel-Interface auf unserer Seite des Tunnels bereitstellt, nennt sich *aiccu*. Wir werden aiccu gleich installieren und konfigurieren. Beim Aufbau des Tunnels und dem Abgleich des Passwortes mit der Gegenseite kommt ein kryptographisches Verfahren zum Einsatz, welches nur dann zu richtigen Ergebnissen kommt, wenn die Uhrzeiten auf beiden Seiten nicht zu stark voneinander abweichen. Deshalb ist es wichtig, vor der Installation von *aiccu* noch einen Server für das *Network Time Protocol* (NTP) zu installieren. Er wird im Hintergrund als Dämon laufen und sich darum kümmern, dass die Systemuhr stets die aktuelle Uhrzeit führt:

Abbildung 3.12 Tunnel-Informationen


```
root@fuzzball:~# apt-get install ntp
✂ After this operation, 1,303 kB of additional disk space ⤸
     will be used.
Do you want to continue [Y/n]? y
✂ Starting NTP server: ntpd.
```

Für das nun folgende Einrichten von aiccu benötigen wir die Zugangsdaten für unseren Tunnel. Das genaue Login des Tunnels ist in den Tunnelinformationen einsehbar (vergleiche Abbildung 3.12).

Dort kann auch ein individuelles, vom Nutzerkonto abweichendes, Passwort für jeden Tunnel hinterlegt werden. Dies ist besonders dann sinnvoll, wenn man mehrere Tunnel beantragt und konfiguriert hat. Da nicht auszuschließen ist, dass eines Tages weitere Tunnel hinzukommen, legen wir ein eigenes Passwort für diesen Tunnel fest. Dann kann es auch schon mit der Installation von aiccu losgehen:

```
root@fuzzball:~# apt-get install aiccu
✂ After this operation, 307 kB of additional disk space ⤸
     will be used.
✂ Setting up aiccu (20070115-14) ...
```

Die restliche Einrichtung des Tunnels beschränkt sich auf das Abarbeiten der Dialoge aus den Abbildungen 3.13 und 3.14. Dabei verwenden Sie natürlich abweichend die Zugangsdaten Ihres eigenen Tunnels.

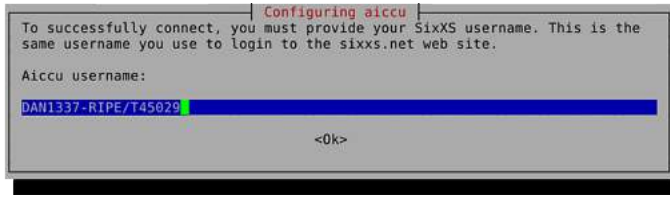


Abbildung 3.13 Dialog zur Abfrage des Tunnel-Logins

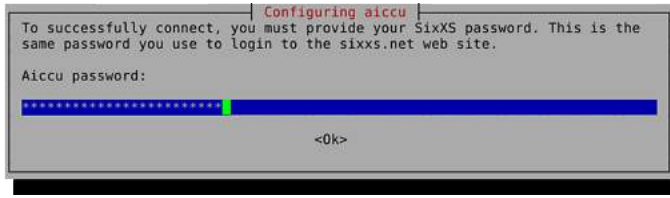


Abbildung 3.14 Dialog zur Abfrage des Tunnel-Passwortes

Bei Problemen mit aiccu hilft ein Blick in das Systemlog weiter:

Probleme mit aiccu

```
user@fuzzball:~$ tail /var/log/syslog
✂ fuzzball kernel: [  965.567085] sit: IPv6 over IPv4 ⤸
      tunneling driver
  fuzzball aiccu[3526]: Response not accepted: Login ⤸
      failed, login/password mismatch.
  fuzzball aiccu[3526]: Couldn't retrieve first tunnel for ⤸
      the above reason, aborting
```

Hier wurde ein Fehler beim Eingeben der Zugangsdaten gemacht, weshalb der Tunnel nicht aufgebaut werden konnte. Derartige Fehler lassen sich beheben, indem die Daten in der Datei `/etc/aiccu.conf` manuell korrigiert werden. Korrekturen werden mit einem Neustart des Programm abgeschlossen, um sie wirksam werden zu lassen:

```
root@fuzzball:~# service aiccu restart
aiccu stop/waiting
aiccu start/running
```

Wenn alles funktioniert hat sollten nun zwei neue Interfaces auf dem System auftauchen:

Erfolgreiche Konfiguration

```
root@fuzzball:~# ip link show
✂ 4: sit0: <NOARP> mtu 1480 qdisc noop state DOWN
      link/sit 0.0.0.0 brd 0.0.0.0
  5: sixxs: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu ⤸
       1280 qdisc pfifo_fast state UNKNOWN qlen 500 link/none
```

Das Interface *sit0* wird von aiccu benötigt und sollte von nun an ignoriert werden. Ganz anders das Interface *sixxs*, dies ist unser Tor in die IPv6-Welt!

3.7 | Global Unicast Addresses

Adressen des Tunnels

Es ist an der Zeit, einen Blick auf das Interface und die ihm zugewiesenen Adressen zu werfen:[9]

```
user@fuzzball:~$ ip addr show dev sixxs
5: sixxs: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu ↩
     1280 qdisc pfifo_fast state UNKNOWN qlen 500
    link/none
    inet6 2a01:198:200:a23::2/64 scope global
       valid_lft forever preferred_lft forever
    inet6 fe80::98:200:a23:2/64 scope link
       valid_lft forever preferred_lft forever
```

Zwei Adressen der Klasse *inet6* aus zwei verschiedenen Address Scopes, von denen wir einen bereits kennen, schmücken das Interface. Von der Link-local Address `fe80::98:200:a23:2` wissen wir bereits, dass sie nur auf dem lokalen Link, hier also innerhalb des Tunnels, Gültigkeit besitzt. Außerhalb des Tunnels ist sie daher nicht weiter von Interesse, wenden wir uns lieber der anderen Adresse zu.

Willkommen im IPv6-Internet!

Der Scope *global* verrät uns, dass die Adresse `2a01:198:200:a23::2` weltweit gültig, und damit auch weltweit eindeutig ist. Herzlichen Glückwunsch, die Maschine *fuzzball* ist nun Teil des weltumspannenden IPv6-Internets! Bereits jetzt wäre es anderen Netzteilnehmern möglich, *fuzzball* Pakete zu schicken.

Fachsprachlich handelt es sich bei `2a01:198:200:a23::2` übrigens um eine Global Unicast Address. Global Unicast Addresses sind vergleichbar mit dem, was man unter IPv4 im Volksmund als *öffentliche IP-Adressen* bezeichnete.

[9] Bitte beachten Sie, dass sich von nun an die im Workshop vorgestellten Adressen von den Ihrigen unterscheiden werden! Ihr Tunnel hat einen eigenen Adressbereich zugewiesen bekommen, der sich von dem des Autors unterscheidet. Vor dem Ausführen von Kommandos müssen Sie im Folgenden stets die gezeigten Adressen durch Ihre eigenen ersetzen!

Präfix	Nutzung	Quelle
0000::/8	Reserviert	RFC 4291 [HD06]
0100::/8	Reserviert	RFC 4291 [HD06]
0200::/7	Reserviert	RFC 4048 [Car05]
0400::/6	Reserviert	RFC 4291 [HD06]
0800::/5	Reserviert	RFC 4291 [HD06]
1000::/4	Reserviert	RFC 4291 [HD06]
2000::/3	Global Unicast	RFC 4291 [HD06]
4000::/3	Reserviert	RFC 4291 [HD06]
6000::/3	Reserviert	RFC 4291 [HD06]
8000::/3	Reserviert	RFC 4291 [HD06]
a000::/3	Reserviert	RFC 4291 [HD06]
c000::/3	Reserviert	RFC 4291 [HD06]
e000::/4	Reserviert	RFC 4291 [HD06]
f000::/5	Reserviert	RFC 4291 [HD06]
f800::/6	Reserviert	RFC 4291 [HD06]
fc00::/7	Unique Local Unicast	RFC 4193 [HH05]
fe00::/9	Reserviert	RFC 4291 [HD06]
fe80::/10	Link-local	RFC 4291 [HD06]
fec0::/10	Reserviert	RFC 3879 [HC04]
ff00::/8	Multicast	RFC 4291 [HD06]

Tabelle 3.3 IPv6-Adressraum

Tabelle 3.3 zeigt, welch kleiner Teil des gesamten IPv6-Adressraumes momentan von den Global Unicast Addresses belegt wird. Sollten sie uns tatsächlich ausgehen, sind noch mehr als genug Reserven vorhanden. Einige kleinere Adressbereiche sind für spezielle Anwendungen reserviert worden (siehe Tabelle 3.4).

Gegenseite im Tunnel

Den Tunnelinformationen bei SixXS entnehmen wir, dass die Gegenseite im Tunnel die Adresse `2a01:198:200:a23::1` hat (Feld *PoP IPv6*). Um die Erreichbarkeit zu überprüfen, verschicken wir Echo Requests an die Gegenseite:

```
user@fuzzball:~$ ping6 -c 3 2a01:198:200:a23::1
PING 2a01:198:200:a23::1 (2a01:198:200:a23::1) 56 data ⤸
     bytes
64 bytes from 2a01:198:200:a23::1: icmp_seq=1 ttl=64 ⤸
     time=273 ms
✂
3 packets transmitted, 3 received, 0% packet loss, time ⤸
     2003ms
```

Tabelle 3.4 Spezielle Adressen und Präfixe

Adresse, Präfix	Nutzung	Quelle
`::1/128`	Loopback	RFC 4291 [HD06]
`::/128`	Unspecified	RFC 4291 [HD06]
`::ffff:0:0/96`	IPv4-mapped	RFC 4291 [HD06]
`64:ff9b::/96`	Übersetzung von IPv6 zu IPv4 und zurück	RFC 6052 [BHB+10]
`0100::/64`	Verworfene Pakete	RFC 6666 [HF12]
`2001:0000::/32`	Teredo Übergangstechnologie	RFC 4380 [Hui06]
`2001:0002::/48`	Benchmarking	RFC 5180 [PHdVD08]
`2001:db8::/32`	Dokumentation	RFC 3849 [HLS04]
`2002::/16`	6to4 Übergangstechnologie	RFC 3056 [CM01]

Die Roundtrip-Zeit, also die Zeit die zwischen dem Absenden eines Paketes und dem Eintreffen der zugehörigen Antwort vergeht, ist gewöhnungsbedürftig. Derart hohe Zeiten sind eigentlich unüblich und in diesem Fall der Tatsache geschuldet, dass mindestens ein (virtueller) SNAT-Router und die gesamte IPv4-Infrastruktur *unter* dem Tunnel liegen. Die Latenz dieser Infrastruktur muss zu der Roundtrip-Zeit eines jeden getunnelten IPv6-Paketes dazugezählt werden. Ein Tunnel taugt also als Lern- und Übergangslösung, für einen langfristigen und produktiven Betrieb ist ein Tunnel aber offensichtlich nicht so gut geeignet.

Traceroute

Je kürzer die Wege im Internet sind, und je besser die Vermaschung der Netze untereinander ist, desto effizienter lässt es sich nutzen. Wie weit der Weg zu einem Ziel ist und wo er (vermutlich) entlangführen wird, lässt sich mit `traceroute6` ermitteln:

```
user@fuzzball:~$ traceroute6 -n trace.ipv6-workshop.de
traceroute to trace.ipv6-workshop.de ⤸
    (2001:67c:26f4:800::6:80) from 2a01:198:200:a23::2, ⤸
    30 hops max, 24 byte packets
 1  2a01:198:200:a23::1  243.071 ms  245.927 ms  262.652 ms
 2  * * *
 3  2a01:198::1  60.541 ms  72.912 ms  71.525 ms
 4  2001:7f8:8::1b1b:0:1  73.614 ms  57.647 ms  66.311 ms
 5  2001:470:0:225::2  50.223 ms  52.88 ms  44.13 ms
 6  2001:7f8:33::a105:7821:1  40.8 ms  38.004 ms  38.649 ms
 7  2001:67c:26f4:800::6:80  64.613 ms  37.025 ms  ⤸
     42.199 ms
```

Ein direkter Nachbar von uns ist der Webserver des IPv6-Workshops im gezeigten Beispiel nicht gerade. Dennoch ist der Weg mit sieben Hops noch verhältnismäßig kurz. Das Lesen von Traceroute6-Ausgaben kann anfangs etwas mühsam sein. Die Adressen sind für Menschen schwieriger zu lesen als die dezimalen Notationen von IPv4.

Welche Schlüsse können Sie aus Ihrem eigenem Traceroute ziehen? Probieren Sie auch andere Ziele aus!

Rundgang im IPv6-Internet

Nehmen Sie sich jetzt ein wenig Zeit das IPv6-Internet zu erkunden. Auf *fuzzball* ist ein Browser installiert, mit ihm können Sie das Web erkunden. Welche Seiten werden bereits über IPv6 ausgeliefert? Sind Ihre Lieblingswebseiten darunter? Nutzen Sie Wireshark um zu überprüfen, welche IP-Version die einzelnen Webseiten nutzen. Sie erkennen die entsprechenden Pakete am einfachsten an den verwendeten Adressen. Alle anderen Informationen können Sie an dieser Stelle noch getrost ignorieren.

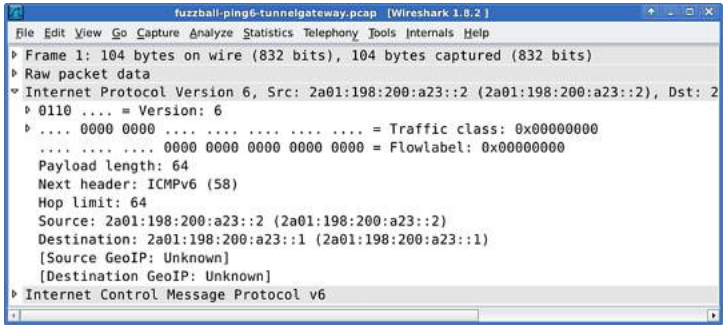
Beliebte Testseiten sind auch *www.kame.net* (die abgebildete Schildkröte sollte tanzen) und *www.test-ipv6.com*.

3.8 | IPv6-Header

IPv6-Header

Wir haben nun schon einige IPv6-Pakete in die Welt geschickt. Was genau wir gesendet haben ist uns aber größtenteils verborgen geblieben. Ein guter Zeitpunkt also um das Gesendete

Abbildung 3.15 IPv6-Header



genauer zu untersuchen. Dazu starten wir Wireshark auf *fuzzball* und lassen das Programm auf dem Tunnelinterface *sixxs* auf *fuzzball* lauschen.

Dann schicken wir wieder ein paar Echo Requests zur Gegenseite des Tunnels:

```
user@fuzzball:~$ ping6 -c 3 2a01:198:200:a23::1
PING 2a01:198:200:a23::1 (2a01:198:200:a23::1) 56 data ⤸
     bytes
64 bytes from 2a01:198:200:a23::1: icmp_seq=1 ttl=64 ⤸
     time=270 ms
✂ 3 packets transmitted, 3 received, 0% packet loss, time ⤸
     2002ms
```

Anschließend stoppen wir das Mitschneiden in Wireshark und suchen uns eines der aufgefangenen Echo Requests aus der Liste aus. In den Paketdetails klappen wir die Zeile *Internet Protocol Version 6* aus und schauen uns den IPv6-Header an. Alternativ kann auch Abbildung 3.15 betrachtet werden.

Wireshark hat uns die Felder des Headers schon menschenlesbar aufbereitet:

Version (4 Bit)
: Die Version des Internet Protocols ist natürlich 6.

Traffic Class (8 Bit)
: Dieses Feld steht der Quelle und den Routern auf der Strecke für die Priorisierung von Paketen zur Verfügung. Der Wert des Feldes darf auf dem Weg von der Quelle zum Ziel geändert werden.

Flow Label (20 Bit)
: Die Quelle kann ein Flow Label setzen um eine eigene Priorisierung festzulegen oder um andere Informationen einzubetten. Das Flow Label darf auf dem Weg von der Quelle zum Ziel zwar ausgewertet, nicht aber verändert werden. Das Feld ist in RFC 6437 [ACJR11] ausführlich beschrieben. Wir können es uns wie eine Art Barcode für Pakete vorstellen.

Payload Length (16 Bit)
: In diesem Feld wird die Länge der Payload in Bytes angegeben. Damit ist die obere Grenze bei knapp 64 KiB.[10]

Next Header (8 Bit)
: Der Next Header gibt an, welcher Header am Anfang der Payload zu erwarten ist. Dabei könnte es sich zum Beispiel um einen TCP-Header handeln. Angegeben wird jeweils die Protokollnummer, daher entsprechen die Werte meist denen, die man auch in IPv4 nutzen würde.

Hop Limit (8 Bit)
: Der Wert dieses Feldes legt die maximale Anzahl an Zwischenstationen beim Routen des Paketes fest. Jeder Router verringert den Wert um Eins. Im IPv4-Header hatte das Feld Time To Live eine vergleichbare Funktion inne. Kommt ein Router dabei auf Null, verwirft er das Paket.

Source Address (128 Bit)
: Das Feld enthält die Adresse der Quelle.

Destination Address (128 Bit)
: Das Feld enthält die Adresse des Ziels.

Bis auf das *Flow Label* kennen wir die Felder so oder in ähnlicher Form schon von IPv4. Dennoch gab es grundlegende Änderungen. Es fehlen altbekannte Felder wie *Fragmentation Offset* und *Checksum* im Header.

[10] Es existiert mit den sogenannten Jumbograms eine Möglichkeit diese Grenze auf 4 GiB anzuheben. In der Praxis ist den Jumbograms aber (noch) keine nennenswerte Bedeutung zugekommen.

Fragmentierung

Während Fragmentierung bei IPv4 noch die Aufgabe aller beteiligten Nodes im Netzwerk war, verlagert sich die Aufgabe bei IPv6 auf die Endpunkte. Ein Router fragmentiert Pakete nicht mehr, die entsprechenden Informationen werden somit nicht im Header benötigt. Sendende Hosts fragmentieren selbstständig wenn ihnen von einem Router mitgeteilt wird, dass ihre Pakete zu groß sind. Die zur Fragmentierung nötigen Steuerinformationen wie Flags und Fragment Offset werden dann in einem sogenannten *Extension Header* ausgelagert. Mit den Extension Headern werden wir uns gleich noch beschäftigen.

Prüfsumme

Überraschen mag auch das Fehlen einer Prüfsumme (Checksum) im Header. Betrachten wir aber die Praxis, dann verwundert es nicht, dass dieses Feld einer schnelleren Verarbeitung in Nodes geopfert wurde. Heutzutage übertragen wir nur äußerst selten IPv6-Pakete über einen Link der nicht schon selbst eine geeignete Prüfsumme enthält. Bei Ethernet, zum Beispiel, ist mit dem *Cyclic Redundancy Check* (CRC) ein schnelles Verfahren im Einsatz, das jedem Frame eine Prüfsumme anhängt. Auch die Schicht über IPv6 bringt in der Regel eigene Prüfsummen mit. Die Protokolle ICMPv6, TCP und UDP nutzen sogar einen Pseudo-Header zur Berechnung derselben, der weite Teile des IPv6-Headers miteinschließt. Eine Prüfsumme ist auf der Internet-Schicht demnach nicht mehr nötig.

Schnelle Verarbeitung in Hardware

Neben dem Verzicht auf Prüfsummenberechnung und Fragmentierung bietet der IPv6-Header noch einen weiteren entscheidenden Vorteil für Router: Er ist immer 40 Bytes lang. Eine variable Länge des Headers, wie wir sie von IPv4 her kennen, bereitet Entwicklern von dedizierter Hardware häufig Kopfschmerzen. Entweder veranschlagt man zu viel oder aber zu wenig Pufferspeicher. Nur selten befindet man sich im optimalen Bereich. Bei IPv6 weiß man stets vorher welcher Anteil der Daten zum Header gehört. Stark vereinfacht würde ein 40 Bytes breites Schieberegister schon reichen, um einen IPv6-Header in Hardware zu speichern und anschließend mit Logikgattern zu verarbeiten.

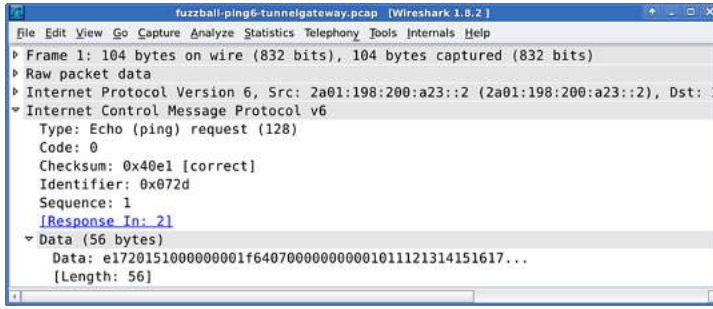


Abbildung 3.16 ICMPv6 Echo Request

3.9 | Internet Control Message Protocol Version 6

Wagen wir einen weiteren Blick in das Paket: Im IPv6-Header stand die Protokollnummer *58* im Feld Next Header. Wireshark war so freundlich uns das schon mit *ICMPv6* zu übersetzen. Folglich steht in der nächsten Zeile in den Paketdetails auch Internet Control Message Protocol v6. Wir klappen diese Zeile aus und erhalten eine Ansicht ähnlich der in Abbildung 3.16.

ICMPv6-Header

Leider können wir der Ansicht noch nicht zweifelsfrei entnehmen, wo der ICMPv6-Header aufhört und die ICMPv6-Nachricht anfängt. Im zugehörigen RFC 4443 [CDG06] finden wir mehr Informationen zum Aufbau des Headers:

Type (8 Bit)
: Im Feld Type steht, um was für eine Nachricht es sich handelt. Anhand des ersten Bits lässt sich bereits eine grobe Einordnung vornehmen. Im Bereich von 0 bis 127 befinden sich die *Error Messages* und ab 128 beginnen die *Informational Messages*. Das Format der Daten nach dem Header hängt vom Typ ab.

Code (8 Bit)
: Das Code-Feld ist abhängig vom Type-Feld. Es gibt genauere Informationen zur Nachricht an, bei Error Messages zum Beispiel den Grund für den Fehler.

Checksum (16 Bit)
: Die Checksum wird über den Inhalt der gesamten ICMPv6-Nachricht und über Teile des vorangestellten

IPv6-Headers gebildet. Neben Quell- und Zieladresse werden die Felder Payload Length und Next Header in die Berechnung einbezogen. Anfangs wird die Prüfsumme auf Null gesetzt, dann findet die Berechnung statt, anschließend wird die berechnete Prüfsumme anstelle der Nullen eingefügt.

ICMPv6-Nachricht

An den Header schließt sich die eigentliche Nachricht an. Ihr Format ist zwar abhängig vom Typ der Nachricht, die Error Messages weisen allerdings eine Gemeinsamkeit auf: Nach dem vierten Byte der Nachricht wird das Paket angefügt, dass die Fehlermeldung provozierte. Das heißt, wir werden bei Error Messages regelmäßig einen IPv6-Header mitten in der Nachricht finden. Aber auch Teile höherschichtiger Protokolle können sich dann in der ICMPv6-Nachricht wiederfinden. Das gesamte Paket darf am Ende jedoch nicht größer als 1280 Bytes werden, gegebenenfalls werden überschüssige Bytes verworfen. 1280 Bytes waren, wir erinnern uns, die minimale Link MTU, die IPv6 voraussetzt. So ist sichergestellt, dass so viele Informationen wie möglich zur Analyse bereitgestellt werden, die Pakete dennoch nicht auf halber Strecke wegen Übergröße verworfen werden.

Echo Request, Echo Reply

Nun sind wir in der Lage zu verstehen was Wireshark uns in den Paketdetails anzeigt. Dem ICMPv6-Header folgen die Felder Identifier und Sequence sowie Data. Der Identifier und die Sequence werden zur Zuordnung von Echo Requests zu Echo Replies genutzt, die Daten sind meist zufällig gewählt.

Schauen Sie sich auch das Antwortpaket (Echo Reply) an, vergleichen Sie dabei die Felder Identifier, Sequence und Data mit denen aus dem Echo Request. Ähneln sich diese?

Informational Messages

Neben Echo Request und Echo Reply existieren noch zahlreiche weitere Informational Messages. Viele dienen der Selbstorganisation der Nodes auf dem lokalen Link, der Ankündigung von Routern und Multicast-Registrierungen. Wir werden sie später noch genauer kennenlernen.

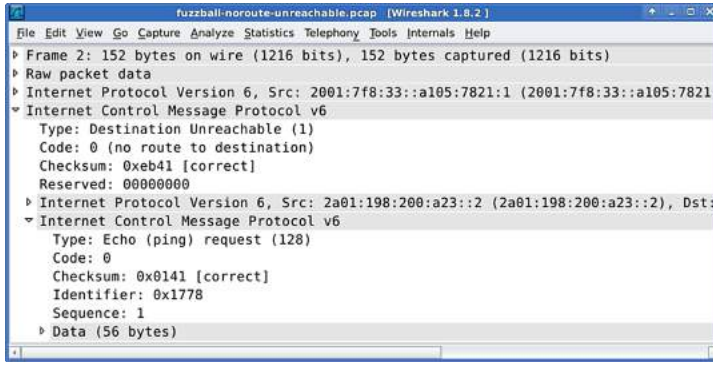


Abbildung 3.17 ICMPv6 Destination Unreachable (No Route)

Destination Unreachable

Nach den Informational Messages werden wir nun Error Messages provozieren um sie einer genaueren Untersuchung zu unterziehen. Wir setzen Wireshark wieder auf das Interface *sixxs* an und schicken Echo Requests an eine Adresse, von der bekannt ist, das sie nicht geroutet wird.

```
user@fuzzball:~$ ping6 -c 3 noroute.ipv6-workshop.de
PING noroute.ipv6-workshop.de (noroute.ipv6-workshop.de) ⤶
     56 data bytes
From 2001:7f8:33::a105:7821:1 icmp_seq=1 Destination ⤶
     unreachable: No route
✂ 3 packets transmitted, 0 received, +3 errors, 100% ⤶
     packet loss, time 2025ms
```

Ein Echo Reply bleibt erwartungsgemäß aus, stattdessen hat der Router mit der Adresse `2001:7f8:33::a105:7821:1` eine Error Message vom Typ Destination Unreachable geschickt. Ein Blick in die Paketdetails in Wireshark verrät uns mehr, alternativ steht Abbildung 3.17 zur Verfügung.

Typ 1 entspricht der Fehlermeldung Destination Unreachable, die Begründung findet sich im Code-Feld. Hier hat es den Wert 0 was *No route to destination* bedeutet. Die Absenderadresse im vorangestellten IPv6-Header ist folgerichtig die Adresse des ersten Routers auf dem Weg, der keine Route zum Ziel mehr kennt. Die eigentliche Nachricht enthält das ursprüngliche Paket. Wireshark nimmt uns wieder eine Menge Arbeit ab und wertet den erkannten IPv6-Header, ICMPv6-Header und den Echo Request aus. Wir können uns

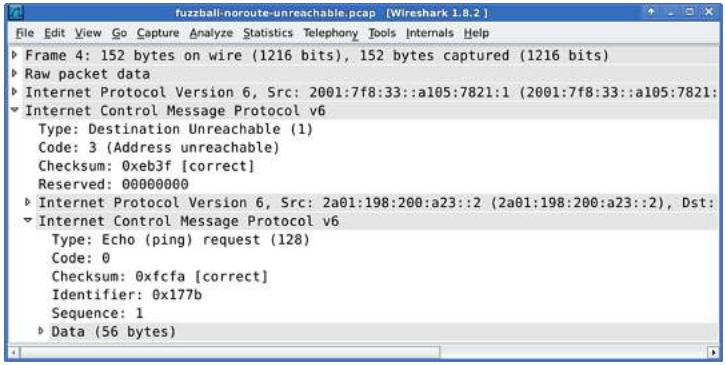


Abbildung 3.18 ICMPv6 Destination Unreachable (Address Unreachable)

bis zum Data-Feld der ursprünglichen Echo Requests durchhangeln.

Wo wir schon dabei sind, provozieren wir gleich noch eine Fehlermeldung. Sobald Wireshark auf Interface *sixxs* startklar ist, schicken wir Echo Requests an eine nicht erreichbare Adresse. Da hier ein paar Timeouts involviert sind, wird es unter Umständen ein wenig dauern bis die ersten Fehlermeldungen eintreffen.

```
user@fuzzball:~$ ping6 -c 3 unreachable.ipv6-workshop.de
PING unreachable.ipv6-workshop.de )
     (unreachable.ipv6-workshop.de) 56 data bytes
From 2001:7f8:33::a105:7821:1 icmp_seq=1 Destination )
     unreachable: Address unreachable
✂ 3 packets transmitted, 0 received, +3 errors, 100% )
     packet loss, time 2490ms
```

Wieder lautet der Fehler Destination Unreachable, aber die Begründung ist diesmal eine andere. Wir analysieren den ICMPv6-Header wieder in Wireshark (siehe Abbildung 3.18).

Im Code-Feld steht diesmal der Wert 3, der für Address Unreachable steht. Es existierte also eine Route zum Ziel, aber auf dem letzten Link war die Adresse nicht zu erreichen. In der eigentlichen Nachricht sehen wir wieder das ursprüngliche Paket.

Error Messages

Die Zahl der Error Messages ist überschaubar, es gibt vier verschiedene Typen. Die Typen 1 und 3 bieten nähere Informationen zum aufgetretenen Fehler im Code-Feld. Um bei einer

eventuellen Fehlerbehebung gut gerüstet zu sein, verschaffen wir uns einen Überblick:

Typ 1: Destination Unreachable
Das Ziel wurde nicht erreicht, eine Begründung befindet sich im Code-Feld.

Code 0: No route to destination
Keine Route zum Ziel.

Code 1: Communication with destination administratively prohibited
Ein Paketfilter hat das Paket verworfen.

Code 2: Beyond scope of source address
Das Ziel befindet sich außerhalb des Gültigkeitsbereiches der Quelladresse. Beispiel: Ein Echo Request von einer Link-local Address an eine Global Unicast Address.

Code 3: Address unreachable
Das Ziel war nicht erreichbar, obwohl eine Route zum Ziel existierte.

Code 4: Port unreachable
Auf dem Ziel war der gewünschte Port nicht erreichbar. Das betrifft Upper-Layer-Protokolle die über Portnummern multiplexen, üblicherweise TCP und UDP.

Code 5: Source address failed ingress or egress policy
Ein Paketfilter hat das Paket aufgrund seiner Absenderadresse verworfen.

Code 6: Reject route to destination
Das Paket wurde verworfen, da die Route zum Ziel nicht gültig war.

Typ 2: Packet Too Big
Das Paket war zu groß für den nächsten Link. Die MTU des betroffenen Links ist in der Fehlermeldung hinterlegt.

Typ 3: Time Exceeded
Das Paket hat seine Lebenszeit überschritten. Näheres kann dem Code-Feld entnommen werden.

Code 0: Hop limit exceeded in transit
Das Hop Limit wurde erreicht und das Paket daher verworfen.

Code 1: Fragment reassembly time exceeded
Das Zusammensetzen der Fragmente auf dem Zielhost dauerte zu lange, das Paket wurde verworfen.

Typ 4: Parameter Problem
Es trat ein Problem beim Verarbeiten eines Feldes im IPv6-Header oder in einem Extension Header auf. Im Code-Feld wird die Ursache, wenn möglich, benannt.

Code 0: Erroneous header field
Das Problem konnte nicht näher eingekreist werden.

Code 1: Unrecognized Next Header type
Das Protokoll, welches im Feld Next Header angegeben war, ist dem Absender der Fehlermeldung nicht bekannt. Er konnte das angegebene Protokoll deshalb nicht auswerten.

Code 2: Unrecognized IPv6 option
In einem Extension Header wurde eine unbekannte Option entdeckt, die Verarbeitung des Extension Headers schlug deshalb fehl.

Path MTU Discovery

Ein Beispiel für die Nützlichkeit der Error Messages zeigt sich bei der Path MTU Discovery. Da unter IPv6 die Router nicht mehr fragmentieren, muss die Quelle die optimale Paketlänge selbst herausfinden. Dazu dienen die Packet Too Big Error Messages der Router als Grundlage. Stellen wir uns ein Szenario, wie in Abbildung 3.19 gezeigt, vor.

Vom Notebook sollen Pakete zum Server gesendet werden. Entlang des Pfades befinden sich zwei Hops, Router A und Router B. Der Link zwischen Router A und Router B kann maximal 1400 Bytes je Frame transportieren, der Link zwischen Router B und dem Server sogar nur 1280 Bytes. Vom Notebook geht das erste Paket mit 1500 Bytes Länge auf die Reise. Schon an Router A wird es, mit der Fehlermeldung Packet Too Big und einem Hinweis auf die MTU von

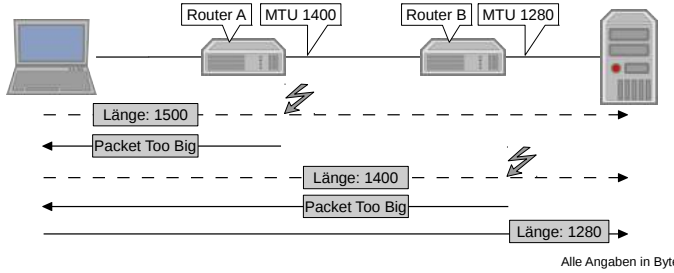


Abbildung 3.19 Path MTU Discovery

1400, zurückgeschickt. Das nächste Paket verschickt das Notebook deshalb mit der neu gelernten Länge von 1400 Bytes. Doch wieder scheitert das Paket an einem der Links auf dem Pfad. Router B verschickt die entsprechende Fehlermeldung und die neue MTU. Das dritte Paket schließlich hat die richtige Länge und gelangt problemlos bis zum Server.

Pfadänderungen

Da die Pfade im Internet einer ständigen Dynamik unterworfen sind, können sich auch die Router und Links auf dem Pfad von einem Endsystem zu einem anderen Endsystem verändern. Deswegen prüfen Hosts regelmäßig, ob die angenommene Path MTU noch aktuell ist. Dazu lassen sie die Pakete langsam größer werden, bis wieder Fehlermeldungen eintreffen.

Bedeutung der Fehlermeldungen

Die Fehlermeldungen von ICMPv6 sind, wie wir gerade gelernt haben, für einen Betrieb von IPv6 unumgänglich. Das blinde Verwerfen aller Fehlermeldungen in einem Paketfilter auf dem Pfad schränkt die Erreichbarkeit der Hosts ein. Paketfilter unter IPv6 müssen demnach mit Bedacht konfiguriert werden. Doch dazu später im Kapitel 7 *Der Paketfilter* mehr.

3.10 | Extension Header

Hintergrund

Der IPv6-Header ist schlank und effizient aufgebaut, nicht zuletzt aufgrund des Verzichts auf unnötige Felder. Die so erzeugten Pakete sehen dann vom schematischen Aufbau her aus wie das Paket A in Abbildung 3.20. Gelegentlich erfordert

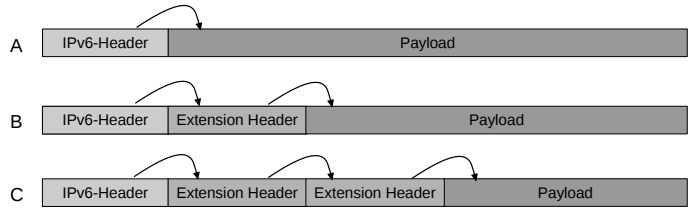


Abbildung 3.20 Extension Header

die Situation aber den Einsatz zusätzlicher Felder. Zum Beispiel bei der Fragmentierung, die zwar nicht mehr von Routern durchgeführt wird, aber nicht grundsätzlich abgeschafft wurde. Hier kommen Extension Header ins Spiel. Sie werden zwischen Header und Payload platziert und enthalten Felder für eine definierte Aufgabe. Der Aufbau entspricht dem von Paket B in Abbildung 3.20.

Verkettung

Im Feld Next Header des IPv6-Headers haben wir bisher den Verweis auf den Header des Upper Layer Protocols gesehen. Befindet sich zwischendrin ein Extension Header, dann verweist Next Header zunächst auf den Extension Header. In jedem Extension Header gibt es wiederum ein Feld Next Header, das auf den Typ des nächsten Headers verweist. Der nächste Header kann entweder ein weiterer Extension Header (Paket C in Abbildung 3.20) oder der Header eines Upper Layer Protocols sein.

Beispiel Fragmentierung

Der Datenblock A in Abbildung 3.21 soll unter Nutzung von IPv6 versendet werden. Da er zu groß ist um als Payload in einem einzelnen IPv6-Paket transportiert zu werden, muss er fragmentiert werden. Die erforderlichen Felder sind im IPv6-Header nicht vorhanden, deshalb kommt der Fragment Extension Header zum Einsatz. Der Datenblock wird geteilt. Offset und Identifikation werden im Fragment Extension Header hinterlegt, dieser wird zwischen IPv6-Header und Payload platziert.

Das Ergebnis sind die Pakete B und C in Abbildung 3.21. Im Ziel wird der Datenblock gemäß den Angaben in den Fragment Extension Headern wieder zusammengesetzt.

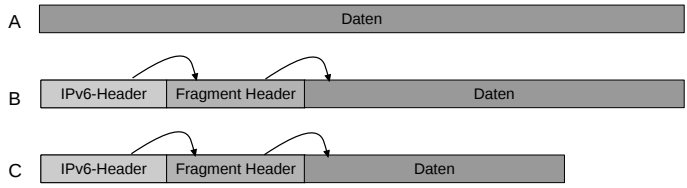


Abbildung 3.21 Extension Header am Beispiel von Fragmentierung

Länge

Die Länge der Extension Header ist stets ein vielfaches von 8 Bytes. Nicht benötigte Bytes werden gegebenenfalls bis zu dieser Grenze aufgefüllt, dieser Prozess wird auch *Padding* genannt. Die Länge in 8-Byte-Schritten ist in allen, auch zukünftigen, Extension Headern an derselben Stelle vermerkt. So kann ein Node einen unbekannten Extension Header einfach überspringen, da er die notwendigen Angaben, Next Header und Länge an bekannter Stelle findet.

Auswertung von Extension Headern

Extension Header werden, von einer Ausnahme abgesehen, vom Ziel verarbeitet. Der Hop-by-Hop Options Extension Header muss auch von den Routern auf dem Pfad ausgewertet werden. Wenn er vorhanden ist, muss er deshalb auch direkt auf den IPv6-Header folgen.

Reihenfolge des Auftretens

Die Reihenfolge der übrigen Extension Header sollte der folgenden Auflistung entsprechen:[11]

- Hop-by-Hop Options
- Destination Options
- Routing
- Shim6
- Fragment
- Authentication Header
- Encapsulating Security Payload
- Destination Options
- Mobility

Extension Header dürfen nur einmal je Paket vorkommen. Auch hier gibt es wieder eine Ausnahme, denn der Destination

[11] Experimentelle Extension Header behandeln wir aus Gründen der Übersichtlichkeit nicht.

Options Extension Header darf in bestimmten Konstellationen zweimal vorkommen.

Parameter Problem

Wenn ein Node einen Extension Header auswerten soll der ihm nicht bekannt ist, dann antwortet er mit einer ICMPv6-Fehlermeldung vom Typ *Parameter Problem*. Die gleiche Meldung soll ein Node verschicken, wenn ein Hop-by-Hop Extension Header vorkommt, dieser aber nicht direkt hinter dem IPv6-Header platziert wurde. Die passenden ICMPv6 Error Messages haben wir schon in Abschnitt 3.9 *Internet Control Message Protocol Version 6* kennengelernt.

No Next Header

Dem Wert 59 im Feld Next Header kommt eine besondere Bedeutung zu. Er zeigt an, dass kein weiterer Header mehr folgt. Das schließt auch Upper Layer Protocols mit ein. Faktisch ist das Paket damit zu Ende, auch wenn noch weitere Bytes folgen sollten. Hosts sind angehalten diesen Überhang zu ignorieren, Router sollen das Paket dennoch unverändert weiterleiten.

Optionen

Einige der Extension Header, die wir gleich besprechen werden, können um eine oder mehrere Optionen ergänzt werden. Dabei werden wir nur jene Optionen betrachten, die in der Praxis am häufigsten anzutreffen sind. Eine vollständige Liste der Optionen wird von der IANA gepflegt und veröffentlicht.[12]

Hop-by-Hop Options und Destination Options Extension Header

Im *Hop-by-Hop Options Extension Header* und im *Destination Options Extension Header* lassen sich diverse Optionen unterbringen. Die Hop-by-Hop Options müssen von allen Nodes auf dem Pfad beachtet werden, die Destination Options nur vom Ziel. Die bekanntesten Optionen sind:

Jumbo Payload

Die *Jumbo Payload Option* erlaubt es, eine Payload zu transportieren, die größer ist als es die Payload Length im IPv6-Header erlauben würde. Die Länge der transportierten Daten wird dann nicht mehr im 16-Bits breiten Feld im Header bestimmt, sondern im 32-Bits breiten Feld in der Jumbo Payload Option. Damit sind Payloads

[12] *http://www.iana.org/assignments/ipv6-parameters/ipv6-parameters.xml*

von knapp unter 4 GiB Länge möglich. Diese Option ist in RFC 2675 [BDH99] beschrieben.

Router Alert

Die *Router Alert Option* gibt Routern einen Hinweis darauf, dass sich im Paket Daten befinden, die auch für den Router von Interesse sein könnten. Jedes Paket zu analysieren würde zu Kapazitätsproblemen und erhöhter Latenz führen. Enthalten jene Pakete, die untersucht werden sollten, jedoch einen Hinweis, so können alle anderen Pakete ohne zeitraubende Prüfung weitergeleitet werden. Genutzt wird die Option beispielsweise beim Beitritt eines Nodes zu einer Multicast Group. Dies ist insbesondere für einen Multicast Router von Interesse. Wir werden dieser Option im Abschnitt 4.5 *Multicast* noch begegnen. In RFC 2711 [PJ99] und RFC 6398 [Fau11] wird die Router Alert Option definiert und ausführlich besprochen.

Optionen besitzen ein Flags-Feld in dem unter anderem festgelegt werden kann, wie ein Node auf eine ihm unbekannte Option reagieren soll. Die Möglichkeiten sind:

- Ignorieren der unbekannten Option und Paket weiterleiten oder verarbeiten.
- Paket verwerfen.
- Paket verwerfen und den Absender darüber informieren.
- Paket verwerfen und den Absender darüber informieren, aber nur wenn die Zieladresse keine Multicast Address ist.

Darüber hinaus kann der Absender den Nodes auf dem Pfad erlauben oder verbieten, die Optionen zu verändern.

Der Hop-by-Hop Options Extension Header wird durch den Wert 0 im Next Header angezeigt, dem Destination Options Extension Header ist der Wert 60 zugeordnet.

Routing Extension Header

Der *Routing Extension Header* hat bereits eine bewegte Geschichte hinter sich. Es gibt drei Typen des Routing Extension Headers, von denen nur einer praktische Bedeutung hat. Der Typ 0 erlaubte es, verbindliche Zwischenziele für ein Paket

zu definieren. Das ermöglichte einfach gestrickte aber hocheffektive Attacken der Kategorie *Denial of Service* (DoS). Mit RFC 5095 [ASNN07] wurde der Typ 0 daher als überholt markiert. Heutzutage wird er von gut konfigurierten Paketfiltern, wie dem in Abschnitt 7.3 *Router-Paketfilter* vorgestellten, verworfen.

Vom Nimrod-Projekt der DARPA wird der Typ 1 benutzt, mehr dazu findet sich in RFC 1992 [CCS96]. Auch dieser Typ wird uns in der Praxis kaum begegnen.

Der Typ 2 schließlich wird von Mobile IPv6 verwendet um die sogenannte *Home Address* eines Nodes zu speichern. Die Wahrscheinlichkeit ist hoch, dass wir mit diesem Typ häufiger konfrontiert werden sobald sich Mobile IPv6 durchgesetzt hat.

Im Feld Next Header wird der Wert 43 genutzt um einen Routing Extension Header anzukündigen.

Shim6 Extension Header

Shim6 ist ein Protokoll zur Failover-Behandlung in Multihomed-Netzen. Es sorgt dafür, dass Verbindungen auch dann noch funktionieren, wenn einer der Provider ausfällt. Fällt ein Präfix aus, werden die Daten mit den Adressen eines anderen Präfixes transportiert. Dem Upper Layer Protocol wird davon aber nichts mitgeteilt, es verhält sich so, als bestünde die ursprüngliche Verbindung noch. Shim6 ist eine spannende Technologie mit noch geringer Verbreitung, ein Blick in das entsprechende RFC 5533 [NB09] lohnt sich trotzdem.

Der Shim6 Extension Header hat den Wert 140 im Feld Next Header.

Fragment Extension Header

Um Pakete zu verschicken welche die Path MTU überschreiten, muss Fragmentierung verwendet werden. Der sendende Node teilt die Payload in Fragmente passender Größe auf. Anschließend erstellt er einen *Fragment Extension Header*, dieser enthält die zur Wiederherstellung der ursprünglichen Payload notwendigen Angaben.

Dem Fragment Extension Header ist der Wert 44 im Feld Next Header zugeordnet.

Authentication Header und Encapsulating Security Payload Extension Header

Die Extension Header *Authentication Header* und *Encapsulating Security Payload* sind Teil der IPsec-Protokollfamilie. Der Authentication Header sichert die Integrität der transportierten Daten und die Authentizität der Quelle. Mit dem Extension Header Encapsulating Security Payload lässt sich die Vertraulichkeit der transportierten Daten durch den Einsatz von Verschlüsselung gewährleisten. Auch die Integrität der transportierten Daten kann sichergestellt werden. Im Gegensatz zum Authentication Header wird dabei nur die Payload und nicht das gesamte Paket berücksichtigt.

Im Next Header steht für Authentication Header der Wert 51 und für Encapsulating Security Payload der Wert 50.

Mobility Extension Header

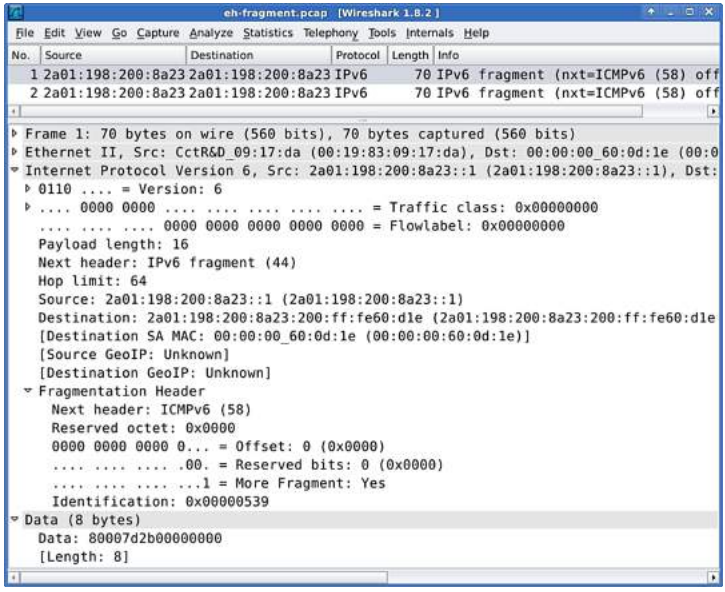
Der *Mobility Extension Header* dient der Verwaltung von mobilen Nodes bei der Nutzung von Mobile IPv6. Dazu enthält ein Mobile Extension Header verschiedene Nachrichten, man könnte auch von einem kleinen Protokoll innerhalb eines Extension Headers sprechen. Zum Beispiel können mobile Nodes ihren Home Router über einen Netzwechsel informieren. In RFC 6275 [PJA11] wird im Mobile Extension Header zwar Platz für einen Next Header reserviert, genutzt wird dieser aber nicht. In der aktuellen Spezifikation folgt einem Mobility Extension Header also kein weiterer Header.

Der Wert 135 im Next Header zeigt einen Mobility Extension Header an.

Beispiel Fragment Extension Header

Nach der Theorie werden wir uns jetzt einen Extension Header in Wireshark anschauen. Dazu greifen wir auf ein praxisnahes Beispiel, die Fragmentierung, zurück. Schematisch haben wir das Beispiel weiter oben bereits besprochen. Das Erzwingen eines Fragment Extension Headers ist nicht ganz einfach, weshalb uns diesmal die Abbildungen 3.22 und 3.23 als Anschauungsmaterial dienen werden. Wenn Sie sich allerdings mit dem Paketmanipulator *Scapy* auskennen, können Sie die

Abbildung 3.22 Fragment Extension Header (1/2)



hier besprochenen Pakete mit dem Quellcode aus Anhang B *Scapy-Skript* selbst erzeugen.

In Abbildung 3.22 sehen wir die ersten acht Bytes eines 16 Bytes langen ICMPv6 Echo Requests. Den zweiten Teil können wir in Abbildung 3.23 betrachten.

Zwischen IPv6-Header und den ICMPv6-Daten steht jeweils ein Fragment Extension Header. Der Wert für Next Header ist daher im IPv6-Header auch 44.

Dass es sich bei den fragmentierten Daten um ICMPv6 handelt, ist im Next Header der Fragment Extension Header vermerkt.

Der *Offset* gibt an, an welcher Stelle der rekonstruierten Payload das jeweilige Fragment einzusetzen ist. Im ersten Paket steht 0x0000 für den Anfang der Payload, das zweite Fragment schließt mit 0x0008 direkt an das erste an. Im zweiten Paket, welches das letzte Fragment transportiert, ist das Flag *More Fragments* nicht mehr gesetzt. Dadurch weiß das Ziel, dass keine weiteren Fragmente mit höherem Offset mehr zu erwarten sind. Beide Fragment Extension Hea-

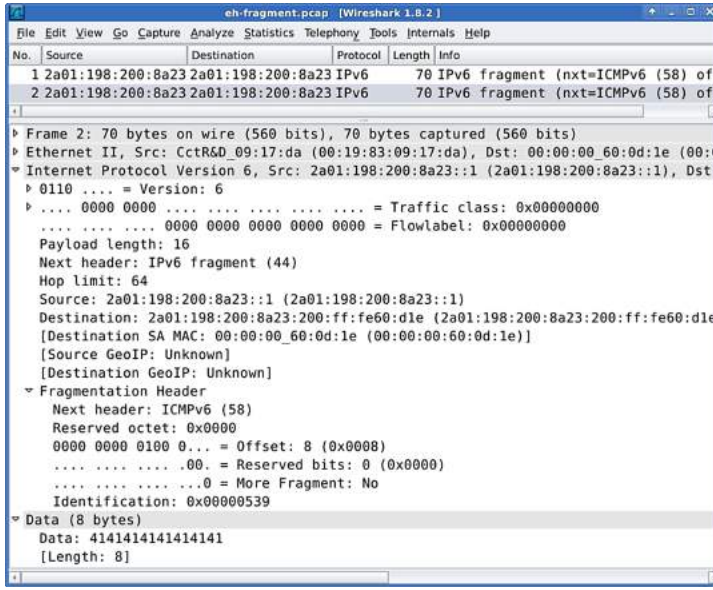


Abbildung 3.23 Fragment Extension Header (2/2)

der weisen den gleichen Wert im Feld *Identification*, nämlich 0x00000539 auf. Damit bilden die beiden gezeigten Fragmente zusammengesetzt die ursprüngliche Payload. Andere Payloads wären auch mit einer anderen Identification versehen.

4 | Die Hosts

Jetzt, wo wir uns eine Maschine gebaut haben die später routen soll, stehen wir vor einem Henne-Ei-Problem. Erweiterten wir *fuzzball* jetzt um die Routing-Funktionalitäten, dann täten wir uns schwer diese zu testen. Uns fehlen im Moment noch die Hosts am internen Link, die den Router benutzen könnten. Wie also sollten wir die Korrektheit des Routings überprüfen? Deshalb werden wir den internen Link zunächst mit Hosts bevölkern. Anschließend wenden wir uns im Kapitel 5 *Der Link* wieder dem Router zu.

Aus lizenzrechtlichen Gründen ist es leider nicht möglich eine Windows-Appliance zur Verfügung zu stellen. Bei der Linux-Appliance haben Sie wieder die Wahl: Sie können die virtuelle Maschine selbst installieren oder als fertige *Appliance* von der Homepage des IPv6-Workshops herunterladen und in VirtualBox importieren.[1] Wenn Sie sich für die Appliance entschieden haben können Sie den folgenden Abschnitt überspringen.

4.1 | Debian GNU/Linux 6

Der erste Host den wir einrichten werden wird auf Debian GNU/Linux 6 basieren.

Konfiguration der Maschine

Der Host soll später am internen Link über IPv6 mit dem Rest der Welt kommunizieren können. Anfangs allerdings, mogeln

[1] http://www.ipv6-workshop.de/

Tabelle 4.1 Parameter der virtuellen Maschine *lynx* während der Installation

Virtuelle Maschine *lynx*	
Betriebssystem	Linux
Version	Debian (64 Bit)
Arbeitsspeicher	512 MB
Festplatte	8.00 GB
Netzwerkadapter 1	NAT MAC-Adresse *000000600d1e*

wir uns um den internen Link noch herum, indem wir den Netzwerkadapter 1 zur Installation an das NAT von VirtualBox anschließen. So erlauben wir der neuen virtuellen Maschine den Zugriff auf das Internet und umgehen den noch nicht fertig konfigurierten Router *fuzzball*. Damit ist garantiert, dass während der Installation Softwarepakete nachgeladen werden können. Tabelle 4.1 zeigt die Konfiguration der neuen virtuellen Maschine während der Installation.

Damit wir mit der Installation des Betriebssystems starten können, legen wir noch das CD-Image von Debian GNU/Linux 6 in das virtuelle CD-Laufwerk der Maschine ein.

Installation des Betriebssystems

Nach dem Einschalten der Maschine begrüßt uns das **Installer boot menu**. Dort starten wir die Installation mit der Auswahl des Menüpunkts **Install**.

Zunächst möchte der Installationsassistent wissen, in welcher Sprache wir die Installation durchführen wollen. Wir wählen die bevorzugte Sprache aus, im Workshop verwenden wir wie gewohnt **English**. Anschließend stellen wir den Ort entsprechend unseres Aufenthaltortes ein, zum Beispiel **Other** →**Europe** →**Germany**. Danach fragt uns der Assistent nach der Zeichenkodierung und dem verwendeten Tastaturlayout. Wir suchen die passenden Einträge aus der Liste heraus und bestätigen sie.

Nach dem automatischen Laden einiger Komponenten verlangt der Assistent wieder nach unserer Aufmerksamkeit. Er möchte das primäre Interface festlegen. Wir bestätigen dazu die Vorauswahl **eth0**, legen als Hostnamen **lynx** fest und lassen den Domainnamen frei.

Im nächsten Schritt vergeben wir das root-Passwort und legen einen unprivilegierten Nutzer an. Als Nutzernamen verwenden wir im Workshop **user**, es steht Ihnen selbstverständlich frei einen eigenen zu wählen. Die Passwörter für den unprivilegierten Nutzer und für root dürfen wir auf keinen Fall vergessen, wir werden Sie noch häufig brauchen.

Mit dem vorgeschlagenen Festplattenlayout **Guided - use entire disk** und der Option **All files in one partition** geben wir uns zufrieden und bestätigen dies in den darauf folgenden Dialogen nochmals.

Das Installationsmedium enthält nur die wichtigsten Softwarepakete und ist deshalb darauf angewiesen, die fehlenden Dateien von einem der offiziellen Server nachzuladen. Die besten Ergebnisse erzielt man üblicherweise mit einem geographisch nahegelegenen Server. Wir wählen also wieder unseren Standort, zum Beispiel **Germany**, aus. Anschließend entscheiden wir uns für einen der angebotenen Server. Sofern keine besonderen Umstände vorliegen, können wir die Proxy-Informationen frei lassen.

Der Assistent installiert nun das Basissystem. Dabei fragt er zwischendurch ab, ob wir damit einverstanden sind am Popularitätswettbewerb teilzunehmen. Eigentlich handelt es sich weniger um einen Wettbewerb als vielmehr um die statistische Erfassung von bevorzugten Softwarepaketen. Wenn Sie das Debian-Projekt unterstützen möchten, können Sie der Teilnahme zustimmen, andernfalls bleiben Sie bei der Vorauswahl **No**.

Im nächsten Schritt, dargestellt in Abbildung 4.1, selektieren wir die gewünschten Software-Komponenten. Das sind einmal das **Graphical desktop environment** und die **Standard system utilities**.

Gegebenenfalls fragt der Assistent noch die eine oder andere Information ab, dabei können wir im Zweifelsfall immer bei der Vorauswahl bleiben. Die letzte Frage die es mit **Yes** zu beantworten gilt, dreht sich um die Installation des Bootloaders.

Abbildung 4.1 Auswahl der Komponenten im Installationsassistenten

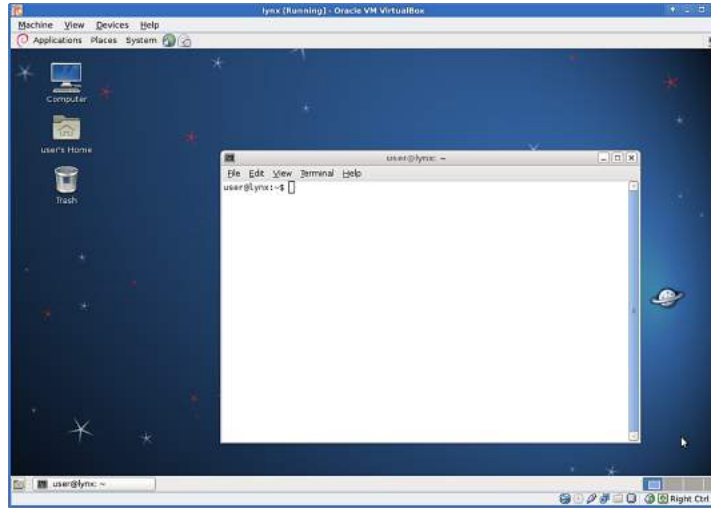


Abbildung 4.2 Desktop mit Terminal nach der Installation

Wenn alles gut gegangen ist, lassen wir mit **Continue** das soeben fertig installierte System starten. Der Desktop, der sich uns nach dem Anmelden präsentiert, sollte wie in Abbildung 4.2 aussehen.

Systemupdate

Im Anschluss führen wir ein Systemupdate durch:

```
root@lynx:~# apt-get update
Get:1 http://ftp.de.debian.org squeeze Release.gpg ↩
    [1,672 B]
✂ Reading package lists... Done
root@lynx:~# apt-get upgrade
Reading package lists... Done
✂ Do you want to continue [Y/n]? y
Processing triggers for menu ...
```

Danach installieren wir Wireshark:

Wireshark installieren

```
root@lynx:~# apt-get install wireshark
Reading package lists... Done
Building dependency tree
✂ Setting up wireshark (1.2.11-6+squeeze9) ...
Processing triggers for menu ...
```

Später werden wir mit Wireshark direkt auf die Interfaces zugreifen. Deshalb erlauben wir den unprivilegierten Nutzern den Zugriff auf die entsprechenden Funktionen:

```
root@lynx:~# setcap cap_net_raw,cap_net_admin=eip ⤸
     /usr/bin/dumpcap
root@lynx:~# getcap /usr/bin/dumpcap
/usr/bin/dumpcap = cap_net_admin,cap_net_raw+eip
```

Wenn der Speicherplatz des Wirtsystems es zulässt, legen wir wieder einen Snapshot an.

Temporäre Anpassungen

Eine wichtige Anpassung müssen wir allerdings noch vornehmen: Der NetworkManager würde uns auch auf dieser Maschine wieder empfindlich beim Arbeiten stören, da er noch für die Konfiguration der Interfaces verantwortlich ist. Die Konfiguration der Interfaces werden wir selbst übernehmen, daher stoppen wir den NetworkManager.

```
root@lynx:~# /etc/init.d/network-manager stop
Stopping network connection manager: NetworkManager.
```

Und damit diese Änderung auch bestehen bleibt, also auch diverse Neustarts überlebt, nehmen wir den NetworkManager aus der Boot-Sequenz heraus.

```
root@lynx:~# update-rc.d network-manager disable
update-rc.d: using dependency based boot sequencing
```

Später im Workshop, wenn wir den NetworkManager benötigen, werden wir ihn wieder aktivieren.

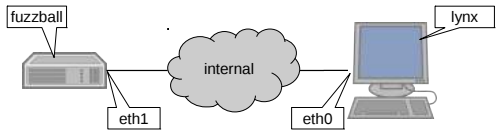
Umzug an den internen Link

Für den Host *lynx* heißt es nun: Umziehen an den internen Link. Der Umzug ist denkbar einfach. Dazu ändern wir lediglich das Netz mit dem der Netzwerkadapter 1 verbunden ist von **NAT** auf **Internal Network** und wählen das Netz **internal** aus. Dies kann zwar prinzipiell problemlos im laufenden Betrieb

Tabelle 4.2 Parameter der virtuellen Maschine *lynx* nach dem Umzug

Virtuelle Maschine *lynx*	
Betriebssystem	Linux
Version	Debian (64 Bit)
Arbeitsspeicher	512 MB
Festplatte	8.00 GB
Netzwerkadapter 1	Internal Network *internal* MAC-Adresse *000000600d1e*

Abbildung 4.3 Link *internal* mit Host *lynx*



durchgeführt werden, dennoch ist ein anschließender Neustart zu empfehlen. Dadurch zwingen wir *lynx* dazu, die Interfaces neu zu initialisieren. Die Parameter von *lynx* entsprechen nun denen aus Tabelle 4.2.

Den Link testen

Wir werfen einen Blick auf das Interface *eth0*, welches jetzt am internen Link angeschlossen ist:

```
user@lynx:~# ip addr show dev eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 ⤸
     qdisc pfifo_fast state UP qlen 1000
    link/ether 00:00:00:60:0d:1e brd ff:ff:ff:ff:ff:ff
    inet6 fe80::200:ff:fe60:d1e/64 scope link
       valid_lft forever preferred_lft forever
```

Das Interface hat bereits eine Link-local Address und das Interface ist hochgefahren. Sollte das Interface aus irgendwelchen Gründen nicht automatisch hochgefahren worden sein, holen wir das nach und lassen uns erneut die Adressen anzeigen. Wie man ein Interface hochfährt haben wir erstmals im Abschnitt 3.1 *Installation des Betriebssystems* behandelt. Zur Erinnerung:

```
root@lynx:~# ip link set up dev eth0
```

Falls *fuzzball* gerade nicht läuft, fahren wir die Maschine jetzt hoch, wir brauchen sie für den folgenden Test. Am Link *internal* befindet sich nun je ein Interface von Router und Host, siehe auch Abbildung 4.3.

Ein Echo Request von *lynx* an die Link-local Address von *fuzzball* soll beweisen, dass am internen Link bereits über IPv6 kommuniziert werden kann.

```
user@lynx:~$ ping6 -c 3 fe80::219:83ff:fe09:17da%eth0
PING fe80::219:83ff:fe09:17da%eth0 ⤸
     (fe80::219:83ff:fe09:17da) 56 data bytes
64 bytes from fe80::219:83ff:fe09:17da: icmp_seq=1 ⤸
     ttl=64 time=0.604 ms
✂ 3 packets transmitted, 3 received, 0% packet loss, time ⤸
     2000ms
```

Wir haben zwei Maschinen auf IP-Schicht (OSI: Schicht 3) miteinander kommunizieren lassen, ohne dass wir selbst Adressen konfigurieren oder Subnetzmasken ausrechnen mussten. Mit IPv4 wären mehr Schritte nötig gewesen.

4.2 | Microsoft Windows 8

Der zweite Host im Workshop wird eine Maschine mit dem Betriebssystem Microsoft Windows 8 sein. Sollten Sie keine passende Lizenz zur Hand haben, können Sie es auch mit der Version 7 probieren, die meisten Kommandos haben sich, im Gegensatz zur Oberfläche, nicht verändert. Steht Ihnen gar keine Lizenz zur Verfügung oder möchten Sie auf den Einsatz proprietärer Betriebssysteme verzichten, sei Ihnen die Installation einer weiteren GNU/Linux-Maschine ans Herz gelegt. Je aktueller die gewählte Distribution dabei ist, desto wahrscheinlicher ist es, dass es Änderungen am IPv6-Stack gab. Wenn sich Nodes mit unterschiedlichen Verhaltensweisen am selben Link befinden, kann dies dem Lerneffekt nur zuträglich sein.

Konfiguration der Maschine

Den Netzwerkadapter 1 schließen wir zur Installation zunächst an das NAT von VirtualBox an. Somit kann die virtuelle Maschine auf das Internet zugreifen, was für Systemupdates unerlässlich ist.

In das virtuelle optische Laufwerk legen wir das Installationsmedium ein, dies kann entweder eine Image-Datei oder ein physisches Laufwerk des Wirtsystems sein. Wählen Sie die

Tabelle 4.3 Parameter der virtuellen Maschine *felis* während der Installation

Virtuelle Maschine *felis*	
Betriebssystem	Windows
Version	Windows 8 (64 Bit)
Arbeitsspeicher	1536 MB
Festplatte	25.00 GB
Netzwerkadapter 1	NAT
	MAC-Adresse *0000000FE715*

für Sie passende Möglichkeit aus und starten Sie die virtuelle Maschine.

Tabelle 4.3 zeigt die Konfiguration der neuen virtuellen Maschine während der Installation.

Installation des Betriebssystems

Nach dem Start fragt uns das Installationsprogramm nach Angaben zur gewünschten Sprache und der Lokalisation. Im Workshop verwenden wir die internationale Version von Microsoft Windows 8. Es steht Ihnen selbstverständlich frei eine lokalisierte Version einzusetzen. Bitte beachten Sie, dass die Bezeichnungen der Schaltflächen dann von denen im Workshop abweichen können. Wir geben die entsprechenden Daten an und betätigen im darauffolgenden Schritt die Schaltfläche **Install now**. Es folgt die Abfrage des Lizenzschlüssels. Danach haben wir die Wahl zwischen einem Upgrade oder einer Neuinstallation. Wir wählen die Neuinstallation, denn die Festplatte der virtuellen Maschine ist noch nicht mit einem Betriebssystem versehen worden. Deswegen übernehmen wir im nächsten Schritt, wo die gesamte Festplatte als Zielmedium angeboten wird, auch die Vorauswahl. Danach kopiert, extrahiert und konfiguriert das Installationsprogramm munter vor sich hin. Erst zur Personalisierung braucht es wieder Hilfe, beispielsweise bei der Auswahl einer Farbkombination und des Computernamens. Während ersteres nicht von Belang ist, legen wir als Computernamen *felis* fest und schreiten voran. Für den Workshop brauchen wir kein Microsoft-Konto, darum nutzen wir die lokale Kontenverwaltung indem wir auf **Local account** klicken. Es folgt eine kleine Demonstration die uns beibringt wie man die Oberfläche am geschicktesten bedient. Wir merken uns, dass man die interessanten Optionen er-

Virtuelle Maschine *felis*	
Betriebssystem	Windows
Version	Windows 8 (64 Bit)
Arbeitsspeicher	1536 MB
Festplatte	25.00 GB
Netzwerkadapter 1	Internal Network *internal* MAC-Adresse *0000000FE715*

Tabelle 4.4 Parameter der virtuellen Maschine *felis* nach der Installation

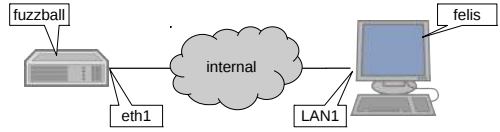


Abbildung 4.4 Netz *internal* mit Host *felis*

reicht, indem man mit der Maus in die rechte, obere Ecke fährt.

Systemupdate

Wir probieren das auch gleich aus, denn im VirtualBox-Fenster eine Ecke zu treffen ohne über sie hinweg zu fahren, erfordert ein wenig Übung. Gelingt uns das kleine Kunststück, erscheint auf der rechten Seite eine Leiste auf der wir **Settings** anklicken. Eine weitere Leiste erscheint, auf dieser wählen wir den untersten Menüpunkt **Change PC settings** aus. Falls noch nicht geschehen, aktivieren wir Windows 8 unter **Activate Windows**. Anschließend geht es mit **Windows Update** weiter unten im Menü weiter. Wir prüfen auf Updates und führen, falls nötig, ein Systemupdate durch. Sobald das Update abgeschlossen ist, kommen wir mit der Windows-Taste wieder in die mit Kacheln dekorierte Ansicht.

Umzug an den internen Link

Der nächste Schritt ist der Umzug von *felis* an den internen Link. Dazu ändern wir das Netz, mit dem der Netzwerkadapter 1 verbunden ist, von **NAT** auf **Internal Network** und wählen das Netz **internal** aus. Die Parameter von *felis* entsprechen nun den in Tabelle 4.4 gezeigten.

Den Link testen

Der Link *internal* sollte nun wie in Abbildung 4.4 dargestellt, aussehen. Es schadet aber auch nicht, wenn *lynx* ebenfalls hochgefahren ist.

Jetzt werden wir den Link testen. Dazu betätigen wir die Tastenkombination Windows-Taste+R und geben dann `cmd` ein. Es

öffnet sich ein Terminal. Im Terminal geben wir als Kommando `ipconfig` ein, um eine Übersicht aller Interfaces und Adressen zu erhalten.

```
C:\Users\user>ipconfig
Ethernet adapter LAN1:
   Link-local IPv6 Address . . . . : ⤸
     fe80::e8a4:145b:625c:e6a1%12
   Autoconfiguration IPv4 Address. : 169.254.230.161
   Subnet Mask . . . . . . . . . . : 255.255.0.0
```

Die Link-local Address hat Windows 8 praktischerweise gleich um das Interface ergänzt, hier Interface *12*. Wir schicken *fuzzball* auf dem lokalen Link einen Echo Request um die Konnektivität zu testen. Das ausgehende Interface entnehmen wir der eben entdeckten Link-local Address und hängen es, zusammen mit einem %-Zeichen, an die Zieladresse an:

```
C:\Users\user>ping -n 3 fe80::219:83ff:fe09:17da%12
Pinging fe80::219:83ff:fe09:17da%12 with 32 bytes of data:
Reply from fe80::219:83ff:fe09:17da%12: time<1ms
    Packets: Sent = 3, Received = 3, Lost = 0 (0% loss),
```

Das Eintreffen von Antwortpaketen zeigt uns, dass der Umzug an den internen Link erfolgreich verlaufen ist.

Übrigens: Gibt es nur ein Interface mit einer Link-local Address, kann man bei `ping` unter Windows 8 die Angabe des Interfaces auch weglassen. Probieren Sie es aus!

4.3 | Neighbor Cache

Am internen Link sind nun drei Nodes vorhanden. Der Router *fuzzball* sowie die Hosts *lynx* und *felis*. Sind alle Maschinen hochgefahren, sieht der interne Link wie in Abbildung 4.5 dargestellt aus.

Neighbor Discovery Protocol

Dank der Link-local Addresses waren die Nodes in der Lage miteinander auf dem internen Link zu kommunizieren. Wir haben dies mit diversen Echo Requests bereits ausprobiert.

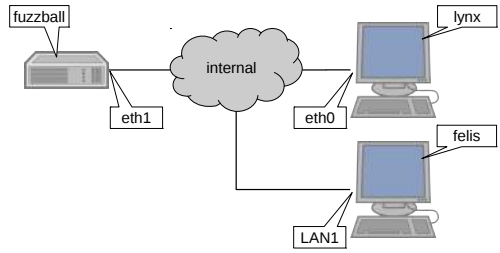


Abbildung 4.5
Link *internal* mit *fuzzball*, *lynx* und *felis*

Das heißt, die Nodes waren in der Lage die Link-local Addresses ihrer Nachbarn zu gültigen Link-layer Addresses aufzulösen. Unser Link *internal* ist ein virtuelles Ethernet, die Link-layer Addresses entsprechen in unserem Netz deshalb den bei Ethernet gebräuchlichen MAC-Adressen. Zuständig für die Auflösung von IPv6-Adressen zu Link-layer Addresses ist das *Neighbor Discovery Protocol* (NDP), spezifiziert in RFC 4861 [NNSS07]. NDP wird über ICMPv6 transportiert. Damit ist es von IPv6 abhängig, wenn es IPv6-Adressen auflösen soll. Wie und warum das dennoch funktioniert, werden wir uns gleich anschauen. Vorher machen wir uns aber noch mit den Neighbor Caches vertraut. Jeder Node betreibt einen Neighbor Cache in dem er die Ergebnisse der Link-layer-Adressauflösungen zwischenspeichert.

Übrigens: Unter IPv4 haben wir für die Auflösung noch ARP benutzt, dessen Ergebnisse in der ARP-Tabelle zwischengespeichert wurden.

Hinweis

Die Einträge in den Neighbor Caches sind je nach Betriebssystem und Zustand unterschiedlich lange gültig. Im Normalfall sind sie sehr kurzlebig. Wir brauchen für die folgenden Experimente deshalb mitunter mehrere Anläufe. Für das Einfangen eines ganz bestimmten Zustands müssen wir zum richtigen Zeitpunkt den Neighbor Cache auslesen. Manchmal ist es daher vorteilhaft, das nächste Kommando schon in der Zwischenablage bereit zu halten, um schneller reagieren zu können.

Neighbor Cache (Windows 8)

Wir wechseln zur Maschine *felis* und öffnen ein Terminal, zum Beispiel mit der bereits bekannten Methode Windows-Taste+R und `cmd`. Um sicher zu gehen, dass sich Einträge im Neighbor

Cache befinden werden, kommunizieren wir mit einem anderen Node auf dem Link. Ein Echo Request an die Link-local Address von *lynx* bietet sich da an.

```
C:\Users\user>ping -n 3 fe80::200:ff:fe60:d1e%12
Pinging fe80::200:ff:fe60:d1e%12 with 32 bytes of data:
Reply from fe80::200:ff:fe60:d1e%12: time<1ms
✂    Packets: Sent = 3, Received = 3, Lost = 0 (0% loss),
```

Sobald eine Antwort eintrifft, können wir den Vorgang abbrechen und uns den Neighbor Cache ausgeben lassen. Das Kommando dazu geben wir unmittelbar nach dem letzten Echo Request ein:

```
C:\Users\user>netsh interface ipv6 show neighbors
✂ Interface 12: LAN1
  Internet Address              Physical Address   Type
  ----------------------------  -----------------  ---------
  fe80::200:ff:fe60:d1e         00-00-00-60-0d-1e  Reachable
  ff02::1                       33-33-00-00-00-01  Permanent
✂ ff02::1:ff60:d1e              33-33-ff-60-0d-1e  Permanent
```

Die IPv6-Adresse `fe80::200:ff:fe60:d1e` wurde korrekt zur Link-layer Address `00-00-00-60-0d-1e` aufgelöst. Der Zustand des Eintrages ist *Reachable*, das heißt, der zugehörige Node gilt auf dem Link als erreichbar. Nach circa 20 Sekunden lassen wir uns den Neighbor Cache ein weiteres Mal ausgeben:

```
C:\Users\user>netsh interface ipv6 show neighbors
✂ Interface 12: LAN1
  Internet Address              Physical Address   Type
  ----------------------------  -----------------  ---------
  fe80::200:ff:fe60:d1e         00-00-00-60-0d-1e  Stale
  ff02::1                       33-33-00-00-00-01  Permanent
✂ ff02::1:ff60:d1e              33-33-ff-60-0d-1e  Permanent
```

Der Eintrag für die Adresse `fe80::200:ff:fe60:d1e` hat seinen Zustand zwischenzeitlich auf *Stale* gewechselt. Die Nachbarn am Link können offensichtlich verschiedene Zustände haben.

Neighbor Cache (Debian GNU/Linux 6)

Die Zustände werden wir in einem weiteren Experiment, diesmal von *lynx* aus, untersuchen. Wir verschicken Echo Requests an *fuzzball*, werden uns aber bei Eingabe der Adresse im letzten Zeichen vertippen. Die Echo Requests gehen an die

Adresse `fe80::219:83ff:fe09:17db`. Den Vorgang brechen wir vorerst nicht ab.

```
user@lynx:~$ ping6 fe80::219:83ff:fe09:17db%eth0
PING fe80::219:83ff:fe09:17db%eth0 ↩
     (fe80::219:83ff:fe09:17db) 56 data bytes
From fe80::200:ff:fe60:d1e icmp_seq=1 Destination ↩
     unreachable: Address unreachable
✂ 48 packets transmitted, 0 received, +42 errors, 100% ↩
     packet loss, time 47182ms
```

Während *lynx* weiter fleißig Echo Requests versendet, öffnen wir ein zweites Terminal und schauen in diesem auf den Neighbor Cache. Unter Linux steht uns dafür das Kommando `ip neighbor show` zur Verfügung.

```
user@lynx:~$ ip neighbor show
fe80::219:83ff:fe09:17db dev eth0  INCOMPLETE
```

Wir lernen mit *Incomplete* einen weiteren Zustand kennen. Nun brechen wir das Senden der Echo Requests im ersten Terminal ab, wechseln zügig in das zweite Terminal und lassen uns erneut den Neighbor Cache ausgeben.

```
user@lynx:~$ ip neighbor show
fe80::219:83ff:fe09:17db dev eth0  FAILED
```

Sofern wir uns nicht zu viel Zeit gelassen haben, sehen wir noch den Zustand *Failed* bevor der Eintrag aus dem Neighbor Cache verschwindet. Failed ist allerdings kein Zustand den man in RFC 4861 [NNSS07] findet, sondern soll verdeutlichen, dass die Auflösung der Adresse nicht erfolgreich beendet werden konnte.

Zustände der Einträge

Den anderen Zuständen hingegen sind klare Bedeutungen zugeordnet:

Incomplete
: Es findet gerade eine Adressauflösung statt. Diese konnte aber noch nicht erfolgreich beendet werden.

Reachable
: Der zugehörige Nachbar gilt als erreichbar. Entweder findet gerade eine Kommunikation mit ihm statt oder eine solche ist erst wenige Sekunden her.

Stale
: Die Erreichbarkeit des zugehörigen Nachbarn ist unbekannt. Es fand kürzlich eine Kommunikation mit ihm statt. Solange keine weitere Kommunikation beabsichtigt ist, sollte auf eine erneute Adressauflösung verzichtet werden.

Delay
: Der zugehörige Nachbar gilt nicht als erreichbar. Es wurde aber wieder eine Kommunikation mit ihm gestartet. Bleibt eine Bestätigung der Erreichbarkeit durch die kommunizierende Schicht aus, wird auf IP-Schicht die Erreichbarkeit aktiv überprüft werden.

Probe
: Der zugehörige Nachbar gilt nicht als erreichbar. Seine Erreichbarkeit wird gerade aktiv geprüft. Sollte die Prüfung fehlschlagen, wird der Eintrag entfernt.

Adressauflösung mitschneiden

Dass die Neighbor Caches funktionieren haben wir gerade gesehen. Auch wissen wir dank der erfolgreichen Echo Requests, dass die Auflösung von IPv6-Adressen und Link-layer Addresses funktioniert. Es ist nun an der Zeit, die Frage nach dem *Wie* zu beantworten.

Dazu suchen wir uns einen Node aus, der gerade einen leeren Neighbor Cache hat, zum Beispiel *fuzzball*. Das Kommando zum Anzeigen des Neighbor Caches ist uns aus den vorherigen Experimenten noch bekannt. Sollten sich noch gültige Einträge im Neighbor Cache befinden, warten wir noch etwas ab. Wenn der Node seine IPv6-Nachbarn schließlich vergessen hat, starten wir Wireshark auf dem entsprechenden Interface, hier *eth1*. Um eine Adressauflösung zu erzwingen versenden wir wieder Echo Requests:

```
user@fuzzball:~$ ping6 -c 3 fe80::200:ff:fe60:d1e%eth1
PING fe80::200:ff:fe60:d1e%eth1 (fe80::200:ff:fe60:d1e) ⤸
     56 data bytes
64 bytes from fe80::200:ff:fe60:d1e: icmp_seq=1 ttl=64 ⤸
     time=3.75 ms
✂ 3 packets transmitted, 3 received, 0% packet loss, time ⤸
     2005ms
```

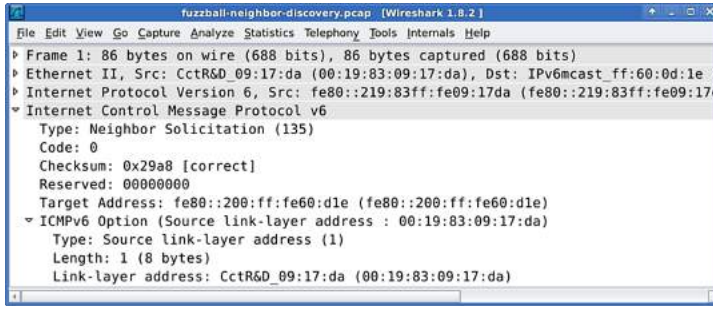


Abbildung 4.6 Neighbor Solicitation

Hat der betroffene Node wie gewünscht geantwortet stoppen wir die Aufzeichnung und schauen uns die Ergebnisse in Wireshark an.

Neighbor Solicitation

Das erste Paket von Interesse ist eine *Neighbor Solicitation* genannte Nachricht (siehe Abbildung 4.6).

Da das Neighbor Discovery Protocol ein in ICMPv6 eingebettetes Protokoll ist, fängt der spannende Teil mit einem ICMPv6-Header an. Der Nachrichtentyp hat die Nummer 135 und den Code 0. Neben der Checksum und reservierten Bytes existiert ein weiteres Feld, die Target Address. Sie enthält die aufzulösende IPv6-Adresse und gehört zum Ziel unseres Echo Requests. Danach folgt eine ICMPv6-Option namens *Source Link-layer Address*. Dabei handelt es sich um zusätzliche Informationen, die der Absender im Paket unterbringen kann. Hier hat *fuzzball* die Link-layer Address seines Interfaces freundlicherweise im Paket vermerkt. Der Gegenseite liefert er damit wertvolle Informationen für ihren eigenen Neighbor Cache. Ganz nebenbei ersparen solche Zusatzinformationen dem Link das ein oder andere Paket.

Solicited Node Multicast Address

Aber wie konnte das Paket überhaupt beim richtigen Node am Link landen, wo wir doch seine Link-layer Address noch nicht kennen? Die Lösung ist im IPv6-Header des Paketes versteckt und lautet Solicited Node Multicast Address. Sie wird wie in Abbildung 4.7 dargestellt berechnet.

Pakete an die Solicited Node Multicast Address eines Nodes können auch bei unbekannter Link-layer Address an diesen

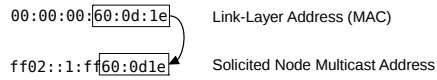


Abbildung 4.7 Solicited Node Multicast Address

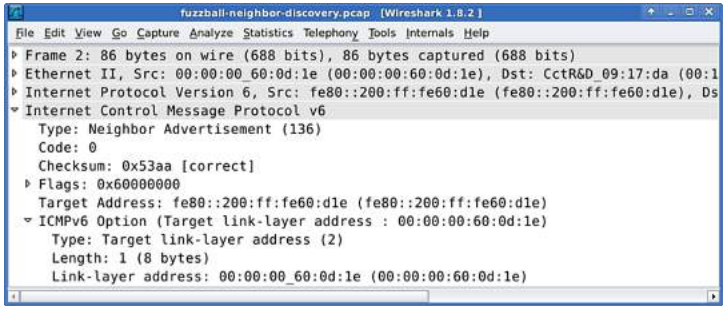


Abbildung 4.8 Neighbor Advertisement

zugestellt werden. Wie genau das funktioniert werden wir in Abschnitt 4.5 *Multicast* lernen. Vorerst werden wir uns damit begnügen, dass es irgendwie funktioniert.

Neighbor Advertisement

Natürlich antwortet der angesprochene Node kurz darauf. Er tut dies mit einem sogenannten *Neighbor Advertisement*. Wieder schauen wir in das Paket, ähnlich Abbildung 4.8, hinein.

Die Typnummer lautet nun 136, der Code bleibt unverändert und hat den Wert 0. Die Target Address gibt weiterhin an, für welche IPv6-Adresse eine Auflösung stattfindet. Neu ist das Feld *Flags*, das wir neugierig aufklappen. Wireshark erläutert die Flags eher sparsam, darum hier ihre Bedeutung:

Router
: Ein gesetztes Flag zeigt an, das es sich bei der Quelle um einen Router handelt. Das ist wichtig zu wissen, da ein Router an einem Link seine Rolle aufgeben kann. Er wird dann zu einem normalen Host. Diese Änderung zeigt er durch das Nichtsetzen des Flags an.

Solicited
: Mit diesem Flag wird darauf hingewiesen, dass das Neighbor Advertisement die Folge einer vorhergehen Neighbor Solicitation ist. Der Empfänger bekommt damit die Erreichbarkeit eines Nodes bestätigt. Mit der

Information kann er seinen Neighbor Cache aktualisieren und Einträge im Zustand Stale oder Probe zurück auf Reachable setzen.

Override
: Mit dem Override-Flag kann die Quelle festlegen, was das Ziel des Neighbor Advertisements in den Neighbor Cache eintragen soll. Ist es gesetzt, werden existierende Einträge zu der angegeben Adresse überschrieben. Ist es nicht gesetzt, darf es lediglich einen unvollständigen Eintrag ergänzen. Adressen, deren Einträge man nicht mit jedem Neighbor Advertisement überschreiben will, sind zum Beispiel Anycast Addresses. Wenn mehrere Anycast Hosts am Link vorhanden sind, laufen die anderen Hosts sonst Gefahr, ständig ihre Neighbor Caches zu überschreiben, ohne einen Nutzen daraus zu ziehen.

Die letzte Information im Paket ist die *Source Link-layer Address*, der eigentliche Grund der Anfrage.

Übrigens: An diesem Beispiel zeigt sich die Flexibilität von IPv6. Die ICMPv6-Optionen werden stets zusammen mit einer Längenangabe versendet. Sollte sich irgendwann eine Link-Technologie durchsetzen die mehr als 6 Bytes für eine Link-layer Address benötigt, steht einer Vergrößerung der entsprechenden ICMPv6-Option nichts im Wege.

Ablauf einer typischen Adressauflösung

Der gesamte Prozess der Adressauflösung zwischen zwei Nodes auf dem Link besteht aus nur zwei Paketen. Von den Paketen musste keines an alle Nodes gesendet werden (siehe Abbildung 4.9). Gegenüber Broadcast stellt das eine deutliche Einsparung dar, gerade an Links mit vielen Nodes.

4.4 | Interface Identifier

Die Nodes am Link *internal* haben sich ihre Link-local Addresses selbst gegeben. Dabei hat jeder Node eine eigene Adresse erhalten, ohne das eine Adresse doppelt vorkam. Bisher haben wir das als gegeben hingenommen oh-

Abbildung 4.9 Ablauf einer typischen Adressauflösung

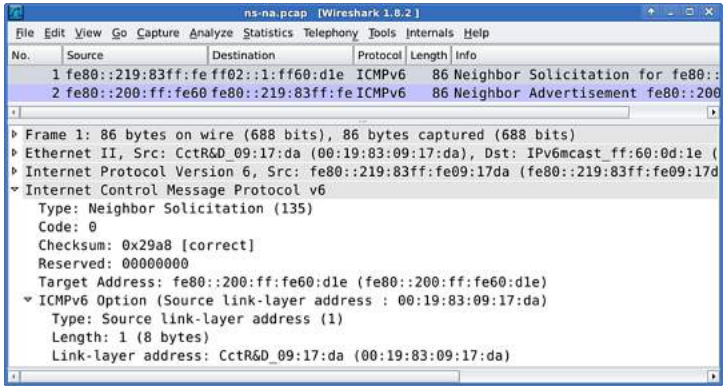
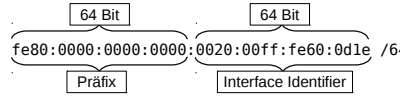


Abbildung 4.10 Link-local Address (ungekürzt)



ne das Prinzip dahinter zu kennen. Das werden wir jetzt ändern.

Interface Identifier

Abbildung 4.10 zeigt eine Link-local Address mit einer Präfixlänge von 64 Bits. Die hinteren 64 Bits sind nicht durch das Präfix vorgegeben und stehen den Interfaces am Link zur Adressierung zur Verfügung. Sie werden deshalb *Interface Identifier* genannt.

Das A und O in IP-Netzen ist die Eindeutigkeit der Adressen innerhalb ihres Scopes.[2] Unter IPv4 haben wir entweder den dezimalen Hostanteil der Adresse mit jedem neuen Node inkrementiert oder die Aufgabe gleich ganz einem DHCP-Server übertragen. Von Tippfehlern oder mutwilligen Störungen abgesehen, war damit die Eindeutigkeit der Adressen gewährleistet. IPv6 nimmt uns das Adressieren des einzelnen Nodes aber ab. Darum muss es auch für die Eindeutigkeit der Adressen Sorge tragen.

[2] An dieser Stelle sprechen wir von Unicast Addresses. Für einen Moment ignorieren wie die Tatsache, dass daneben noch Multicast und Anycast Addresses existieren.

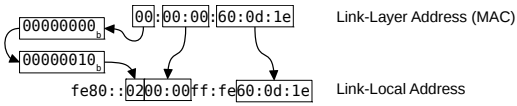


Abbildung 4.11 Bildung der Link-local Address von *lynx*

Von der Mac-Adresse zum EUI-64

Was also liegt näher, als die Link-layer Address des zugehörigen Interfaces als Quelle für einen eindeutigen Interface Identifier herzunehmen? Benutzt das Interface schon einen *Extended Unique Identifier* mit einer Länge von 64 Bits, dann kann dieser ohne Umwege übernommen werden.[3] Auf dem klassischen Ethernet sind die Link-layer Addresses aber nur 48 Bits lang.

Schauen wir doch bei *lynx* nach, wie sein IPv6-Stack das Problem gelöst hat. Als erstes ein Blick auf die Link-layer Address:

```
user@lynx:~$ ip link show dev eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 ⤸
     qdisc pfifo_fast state UP qlen 1000
    link/ether 00:00:00:60:0d:1e brd ff:ff:ff:ff:ff:ff
```

Danach die Link-local Address des Interfaces:

```
user@lynx:~$ ip addr show dev eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 ⤸
     qdisc pfifo_fast state UP qlen 1000
    link/ether 00:00:00:60:0d:1e brd ff:ff:ff:ff:ff:ff
    inet6 fe80::200:ff:fe60:d1e/64 scope link
       valid_lft forever preferred_lft forever
```

Offensichtlich wurde die Link-layer Address in den Interface Identifier aufgenommen. Der Stack hat das zweit-niederwertigste Bit im ersten Byte der Link-layer Address gekippt. In der Mitte der Link-layer Address wurden darüberhinaus die hexadezimale Zeichenfolge `ff:fe` eingefügt. Auf diese Weise ist ein sogenannter *Modified EUI-64* entstanden. In Abbildung 4.11 ist der Vorgang graphisch dargestellt.

[3] 64-Bit Extended Unique Identifier haben, ebenso wie 48-Bit Extended Unique Identifier (auch bekannt als MAC-Adressen), die Aufgabe, auf dem Link-layer teilnehmende Geräte zu adressieren. Um einer Knappheit an EUI-48 frühzeitig vorzubeugen, hat die IEEE die EUI-64 definiert.

Bitspielereien

Das gekippte Bit verwirrt im ersten Augenblick. Die Erklärung liegt in der Bedeutung des betroffenen Bits verborgen, dessen Name *Universal/Local Bit* lautet. Es zeigt an, ob die MAC-Adresse vom Hersteller vergeben wurde (Wert 0), oder ob sie lokal überschrieben wurde (Wert 1). Die Werte 0 und 1 des Universal/Local Bit sind bei einem Modified EUI-64 von umgekehrter Bedeutung, deswegen verlangt der IPv6-Standard das Kippen desselben.

Privacy Extensions

Der so generierte Interface Identifier findet sich nicht nur in Link-local Addresses, sondern auch in Global Unicast Addresses wieder. Die Link-layer Adress eines Nodes, dessen IPv6-Adresse einen solchen Interface Identifier enthält, kann zurückgerechnet werden. Das beeinträchtigt natürlich die Privatsphäre der betroffenen Nutzer. Erstens können Kommunikationspartner einen Node wiedererkennen, selbst wenn er zwischenzeitlich das Präfix gewechselt haben sollte, denn der Interface Identifier ändert sich bei einem Präfixwechsel nicht. Zweitens lässt sich vom Interface Identifier auf den Hersteller des verwendeten Interfaces schließen. Damit ist einem umfangreichen Tracking und Profiling von Nutzern Tür und Tor geöffnet.

Das Problem wurde schließlich mit der Einführung der *Privacy Extensions* gelöst. Das sind Adressen, die keine Rückschlüsse mehr auf die Link-layer Address des Interfaces zulassen. Windows 8 geht sogar so weit, und setzt die Privacy Extensions auch bei den Link-local Addresses ein. Das werden wir uns auf *felis* ansehen:

```
C:\Users\user>ipconfig
Ethernet adapter LAN1:
✂   Link-local IPv6 Address . . . . . : ↩
      fe80::e8a4:145b:625c:e6a1%12
```

Ein Muster ist nicht zu erkennen, und auch das markante `ff:fe` fehlt. Diese Adresse folgt keinem bestimmten Muster! Es handelt sich um eine zufällige Folge hexadezimaler Zeichen. In regelmäßigen Abständen wird eine neue, zufällige Adresse generiert und die alte verworfen. Die Standardeinstellung vieler

Betriebssysteme dafür ist 24 Stunden. Ein Tracking des Nutzers ist so, zumindest auf IP-Schicht, nicht über 24 Stunden hinweg möglich.

Die zufällig generierten Adressen kann man Windows 8 mit `netsh` abgewöhnen. Für die Änderung dieser Einstellung verlangt Windows 8 Administratorrechte, darum führen wir das folgende Kommando als Administrator aus. Dazu betätigen wir die Tastenkombination Windowstaste+X. Es erscheint ein Menü in dem wir den Punkt *Command Prompt (Admin)* auswählen:

```
C:\Users\user>netsh interface ipv6 set global ⤸
    randomizeidentifiers=disabled
Ok.
```

Von der Wirkung können wir uns sofort überzeugen.

```
C:\Users\user>ipconfig /all
Ethernet adapter LAN1:
✂   Physical Address. . . . . . . . . : 00-00-00-0F-E7-15
✂   Link-local IPv6 Address . . . . . : ⤸
    fe80::200:ff:fe0f:e715%12(Preferred)
```

Diesmal wurde die Adresse mit Hilfe eines Modified EUI-64 erzeugt. Wenn Sie möchten, können Sie die Privacy Extensions jetzt wieder einschalten:

```
C:\Users\user>netsh interface ipv6 set global ⤸
    randomizeidentifiers=enabled
Ok.
```

4.5 | Multicast

Wir haben beim ersten Blick auf die *Neighbor Solicitation* eine wichtige Frage vorerst zurückgestellt: Wie verschicken wir Pakete an Ziele, deren Link-layer Address uns noch nicht bekannt ist? Die Lösung heißt Multicast und die zugehörigen Adressen sind die *Multicast Addresses*. Multicast Addresses sind in der Lage, mehrere Interfaces anzusprechen. Besonders nützlich ist das, wenn sich die angesprochenen Interfaces in verschiedenen Nodes befinden. So können wir indirekt Gruppen von

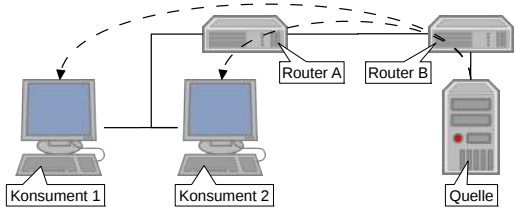


Abbildung 4.12 Übertragung eines Datenstroms ohne Multicast

Nodes ansprechen. Alle Interfaces, die innerhalb eines Gültigkeitsbereiches dieselbe Multicast Address nutzen, bilden deshalb eine *Multicast Group*.

Anwendungsgebiete von Multicast

In IPv6 verwenden wir Multicast hauptsächlich zur

- Vermeidung von mehrfacher Übertragung identischer Datenströme und
- effizienten Organisation der Nodes an einem Link.

Szenario

Um die Nutzung von Multicast bei Datenströmen zu verstehen, denken wir uns in folgende Situation hinein: Mehrere Zuschauer möchten die Live-Übertragung einer Sportveranstaltung als Videostream auf ihren IPv6-fähigen Endgeräten verfolgen. Das können Computer, Smartphones oder netzwerkfähige Fernseher sein. Das Videomaterial wird in Form eines Datenstroms von einem Server bereitgestellt, den wir *Multicast-Quelle* nennen. Auf dem Weg zu den Zuschauern, im Folgenden *Konsumenten* genannt, passiert der Datenstrom zwei Router. Da es sich um eine Live-Übertragung handelt, erhalten alle Konsumenten zu jedem Zeitpunkt die gleichen Daten. Wird kein Multicast verwendet, dann wird nach dem in Abbildung 4.12 dargestellten Prinzip verfahren.

Jeder Konsument kontaktiert die Quelle des Datenstroms und erhält von ihr die Daten, obwohl weitgehend identisch, individuell zugestellt. Alle beteiligten Links und Router auf dem Pfad werden mehrfach belastet. Mit jedem neuen Konsumenten erhöht sich das Datenaufkommen.

Multicast-Datenströme

Nehmen wir an, die Sportveranstaltung aus unserem Beispiel gewinnt an Popularität. Plötzlich sind wir mit einer hohen Nachfrage konfrontiert. Das bisherige Vorgehen ist nicht

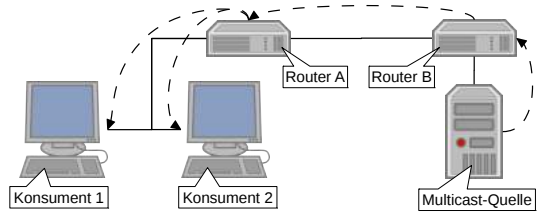


Abbildung 4.13 Übertragung eines Datenstroms mit Multicast

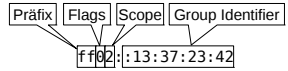


Abbildung 4.14 Multicast Address

in der Lage, diese Nachfrage zu befriedigen, da die Kapazitätsgrenzen der Links und Router erreicht werden würden. Eine Lösung stellt Multicast dar. Abbildung 4.13 zeigt das Verfahren, die beteiligten Systeme bleiben die gleichen wie vorher.

Der wesentliche Unterschied ist, dass nun nicht mehr die Quelle für die Duplizierung des Datenstroms zuständig ist, sondern diese Aufgabe den Routern zuteilwird. Der Datenstrom wird genau einmal von der Multicast-Quelle zu Router B übertragen, dieser wiederum sendet die Pakete genau einmal an Router A weiter. Von Router A schließlich wird der Datenstrom an die Konsumenten verteilt. Je besser die Hardware am Link (zum Beispiel ein Switch) mit Multicast umgehen kann, desto später findet die technische Duplizierung des Datenstroms statt. Idealerweise wird bei Nutzung von Ethernet erst am letzten Switchport eine Kopie der betroffenen Frames angefertigt.

Multicast Addresses

Multicast Addresses sind durch das Präfix `ff::/8` gekennzeichnet. Sie beginnen also stets mit `ff`, daran können wir sie schnell erkennen. Danach folgen 4 Bits für Flags und weitere 4 Bits für die Angabe des *Multicast Scopes*. Der grundlegende Aufbau einer Multicast Address ist in Abbildung 4.14 dargestellt.

Die Flags (in Leserichtung) haben folgende Bedeutung:

Reserviert
: Das erste Flag ist reserviert für spätere Erweiterungen. Es wird auf Null gesetzt.

Rendezvous Point

Das *Rendezvous Point Flag* wird gesetzt wenn im Group Identifier die Adresse eines Rendezvous Points eingebettet ist. Das Verfahren wird für *Inter Domain Multicast* verwendet, ein Thema das außerhalb des Workshops liegt. Interessierte finden mehr dazu im RFC 3956 [SH04].

Prefix

Mit dem *Prefix Flag* wird angezeigt, dass im Group Identifier ein Präfix und die zugehörige Präfixlänge eingebettet wurden. Nodes können das Präfix extrahieren und so erfahren, in welchem Netz die Multicast-Quelle beheimatet ist. Weitere Informationen zur Verwendung des Flags hält RFC 3306 [HT02] bereit.

Transient

Ein gesetztes *Transient Flag* bedeutet, dass es sich nicht um eine Well-known Multicast Address handelt. Letztere werden von der IANA öffentlich verwaltet.[4] Tabelle 4.5 enthält eine Auswahl üblicher Well-known Multicast Addresses.

Multicast Scopes

Während einige Multicast Scopes fest definierte Grenzen haben, lassen sich andere individuell festlegen. Die Einhaltung der Grenzen kann dann nur gewährleistet werden, wenn die unternehmenseigenen Router entsprechend konfiguriert wurden. Mehr Informationen zu den Multicast Scopes können der Tabelle 4.6 entnommen werden.

Organisation von Multicast Groups

Eine Multicast Address identifiziert eine Multicast Group innerhalb ihres Scopes eindeutig. Für die Organisation dieser Gruppen ist das Protokoll *Multicast Listener Discovery v2* (MLDv2) zuständig. Es ist in RFC 3810 [VC04] und RFC 4604 [HCH06] standardisiert und definiert verschiedene Nachrichten, welche über ICMPv6 transportiert werden. Die heute gebräuchlichsten Nachrichten sind die

- *Multicast Listener Query Message* und die
- *Multicast Listener Report Message*.

[4] *http://www.iana.org/assignments/multicast-addresses*

Scope: Node	
Adresse, Präfix	**Nutzung**
`ff01::1`	Alle Nodes
`ff01::2`	Alle Router
`ff01::fb`	Multicast-DNS
Scope: Link	
Adresse, Präfix	**Nutzung**
`ff02::1`	Alle Nodes
`ff02::2`	Alle Router
`ff02::b`	Mobile Agents (Mobile IPv6)
`ff02::f`	Universal Plug and Play
`ff02::16`	Alle MLDv2-fähigen Router
`ff02::6a`	Alle Multicast-Abonnenten
`ff02::fb`	Multicast-DNS
`ff02::1:2`	Alle DHCP-Agents
`ff02::1:ff00:0/104`	Solicited Node Multicast Address
Scope: Site	
Adresse, Präfix	**Nutzung**
`ff05::2`	Alle Router
`ff05::1:3`	Alle DHCP-Server
`ff05::fb`	Multicast-DNS
Scope: Variabel (X)	
Adresse, Präfix	**Nutzung**
`ff0X::fb`	Multicast-DNS
`ff0X::101`	NTP
`ff0X::db8:0:0/96`	Dokumentation

Tabelle 4.5 Well-known Multicast Addresses (Auswahl)

Tabelle 4.6 Multicast Scopes

Scope	Name	Beschreibung
`0x0`	Reserviert	
`0x1`	Interface-Local	Das Multicast-Äquivalent zur Loopback Address `::1`. Nur auf dem lokalen Node gültig.
`0x2`	Link-local	Nur auf dem Link gültig. Die Pakete dürfen nicht geroutet werden.
`0x3`	Reserviert	
`0x4`	Admin-Local	Vom Administrator festzulegender Teil des Netzes. Wird auch für eigenständig administrierte Abteilungen verwendet.
`0x5`	Site-Local	Lokales, physikalisches Netzwerk. In der Regel eine Filiale oder eine Betriebsstätte.
`0x6-0x7`	Nutzerdefiniert	Zur freien Definition eigener Gültigkeitsbereiche.
`0x8`	Organization-Local	Ein die gesamte Organisation umfassender Gültigkeitsbereich. Umfasst auch geographisch entfernte Betriebsstätten, die über Standleitung oder *Virtual Private Network* (VPN) angebunden sind.
`0x9-0xd`	Nutzerdefiniert	Zur freien Definition eigener Gültigkeitsbereiche.
`0xe`	Global	Das gesamte Internet.
`0xf`	Reserviert	

Darüber hinaus müssen Nodes aus Kompatibilitätsgründen auch die älteren MLDv1-Nachrichten

- *Version 1 Multicast Listener Report* und
- *Version 1 Multicast Listener Done*

aus RFC 2710 [DFH99] unterstützen.

Listener Query Message

Mit der Listener Query Message kann festgestellt werden, ob eine Gruppe noch Mitglieder hat. Damit kann auch ein nicht ordnungsgemäßes Verlassen einer Gruppe bemerkt werden.

Listener Report Message

Die Listener Report Messages dienen dazu, Multicast Groups beizutreten oder sie zu verlassen. Jedes Interface, das Mitglied einer bestimmten Gruppe ist, nimmt für diese Gruppe die Rolle eines *Multicast Listeners* ein. Pakete die an die Gruppenadresse gesendet wurden, werden allen Mitgliedern zugestellt. Möchte ein Interface für eine Gruppe keine Pakete mehr erhalten, so verlässt es die Gruppe wieder.

Gruppenbeitritt mitschneiden

Den Beitritt zu einer Gruppe werden wir uns in Wireshark anschauen. Dazu fahren wir das Interface *eth0* auf *lynx* zuerst herunter:

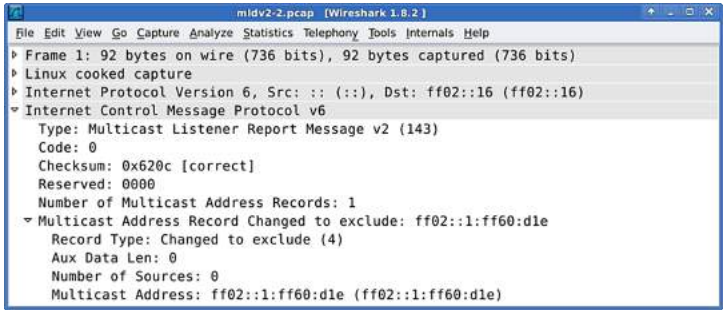
```
root@lynx:~# ip link set down dev eth0
```

Jetzt starten wir Wireshark und lassen ihn auf dem *Pseudo-Interface* lauschen. Das Pseudo-Interface liefert die Daten von allen Interfaces, auch von denen, die erst nach dem Start des Mitschnitts hochgefahren werden. Wir haben also die Chance, den Start eines Interfaces zu beobachten. In diesem Zeitraum treten nämlich vermehrt MLDv2 Messages auf. Dazu fahren wir das Interface wieder hoch:

```
root@lynx:~# ip link set up dev eth0
```

Von den zahlreichen Paketen die von Wireshark mitgeschnitten wurden suchen wir uns eines vom Typ MLDv2 aus. Es sollte dem in Abbildung 4.15 gezeigten ähneln.

Abbildung 4.15 Multicast Listener Discovery



Wir sehen einen Multicast Listener Report, eingebettet in ICMPv6 als Typ 143. Neben den bekannten Code-, Checksum- und Reserved-Feldern ist auch ein Feld namens *Number of Multicast Address Records* vorhanden. Es gibt an, wie viele *Multicast Address Record Changes* folgen. In unserem Beispiel folgt nur ein Eintrag, es wäre auch möglich mehrere Einträge auf einmal bekanntzugeben. Jeder Eintrag steht für Änderungen der Zugehörigkeit zu Multicast Groups eines Interfaces.

Include und Exclude

Die Interpretation der Einträge ist nicht ganz trivial. Es gibt zwei Arten von Multicast Listener Report Messages. Diese heißen *Include* und *Exclude*. Ein Include steht allerdings nicht, wie der Wortlaut vielleicht vermuten lässt, für einen Gruppenbeitritt eines Interfaces. Und ein Exclude muss nicht unbedingt einen Gruppenaustritt bedeuten. Include und Exclude beziehen sich nämlich nicht auf die Multicast Group, sondern auf eine Liste von Quellen, von denen ein Interface Pakete an die Gruppe akzeptiert (Include) oder auch nicht akzeptiert (Exclude). Nun kann ein Interface auch eine leere Liste mitschicken, und damit anzeigen, dass sich das Include oder Exclude auf keine Adressen bezieht. Dann käme ein Include einem Gruppaustritt gleich, und ein Exclude mit leerer Liste entspräche einem Gruppenbeitritt. Multicast Listener Report Messages erwecken manchmal den Eindruck einer doppelten Verneinung.

Beispiel: Gruppenbeitritt

Dazu als Beispiel unsere MLDv2 Message: Wir klappen den Eintrag *Multicast Address Record Changed to exclude* auf und sehen uns die enthaltenen Informationen an. Wir sehen die

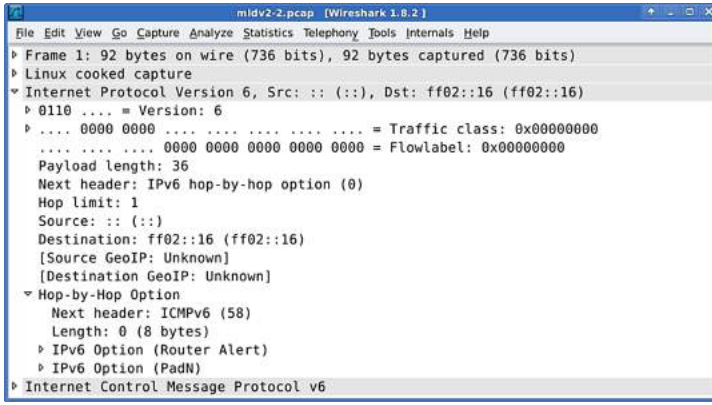


Abbildung 4.16 IPv6-Header eines MLDv2-Paketes

Art der Änderung, Exclude, und weitere Felder. Das Feld *Aux Data Length* gibt an wie viele 8-Byte-Blöcke mit zusätzlichen Daten der Multicast Address folgen. Da es hier den Wert 0 hat, sind keine weiteren Daten zu erwarten. Es folgt die Anzahl der einschränkenden Quellen, hier 0, und im Anschluss die betroffene Multicast Address.

Nach Auswertung aller Felder kommen wir zum dem Schluss: Es handelt sich in unserem Beispiel um eine Änderung der Art Exclude ohne Einschränkung der Quellen. Für diese Multicast Group akzeptiert das Interface Pakete von allen Quellen, außer denen in der Liste. Da die Liste leer ist, gibt es keine Quellen die vom Interface keine Pakete akzeptieren würde. Es handelt sich also um einen uneingeschränkten Gruppenbeitritt.

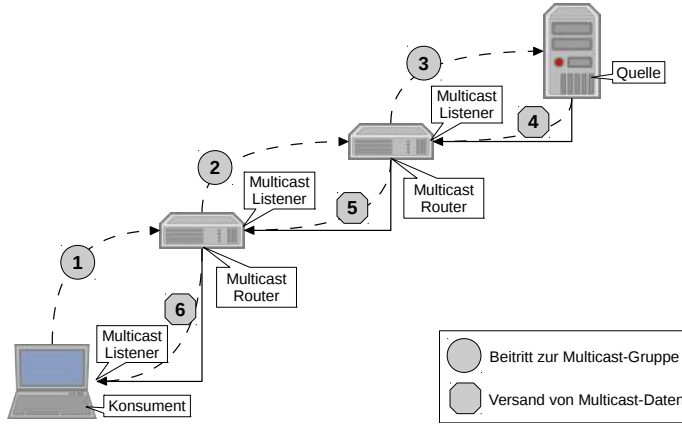
Hop Limit

Abschließend schauen wir uns noch das Feld Hop Limit des vorangestellten IPv6-Headers an (siehe Abbildung 4.16). Es hat den Wert 1, was bedeutet, dass MLDv2-Pakete einen Router nicht passieren können. Die ganze Organisation von Multicast Groups findet also auf dem lokalen Link statt.

Multicast Routing

Trotzdem möchte ein Interface vielleicht den Datenstrom einer Multicast-Quelle empfangen, die an einem anderen Link angeschlossen ist. Wenn die Pakete den Router nicht passieren können, muss der Router anderweitig tätig werden, um den angeforderten Datenstrom bereitzustellen. Dazu muss er

Abbildung 4.17
Multicast Routing



Multicast-fähig sein. Er wird dann auch *Multicast Router* genannt. Tatsächlich, wenn wir den Header weiter analysieren, entdecken wir einen Hop-by-Hop Extension Header mit Router Alert Option. Der Router wird explizit darauf hingewiesen, dass dieses Paket für ihn wichtige Informationen enthalten könnte. Um den Datenstrom auf dem anfragenden Link bereitzustellen, muss der Router seinerseits den Datenstrom anfordern. Entweder hat er ein Interface an einem Link an dem auch die Multicast-Quelle angeschlossen ist, dann tritt er einfach der entsprechenden Gruppe bei. So erhält er die zugehörigen Pakete die er dann zum anfragenden Link routen kann. Hat der Router kein Interface am Link der Multicast-Quelle, beauftragt er einen der ihm bekannten Multicast Router, die Daten bereitzustellen. Dazu tritt er wieder der Multicast Group bei und nimmt für diese Gruppe am betroffenen Interface die Rolle eines Multicast Listeners ein. Das Prinzip ist beispielhaft in Abbildung 4.17 dargestellt, hier ist die Quelle zwei Hops vom Konsumenten entfernt.

4.6 | Multicast auf dem Link-layer

All Nodes Multicast Address

Wo bei IPv4 häufig Broadcast zum Einsatz kam, wird unter IPv6 Multicast verwendet. Immer dann, wenn nicht alle Nodes am Link angesprochen werden sollen, ist die Verwendung von

Multicast ressourcenschonender. Idealerweise werden nur jene Nodes behelligt, die auch Interesse an den versendeten Daten haben. Sollen doch einmal alle Nodes eines Links angesprochen werden, kann die All Nodes Multicast Address mit Link-local Scope genutzt werden. Sie repräsentiert eine Multicast Group der alle Nodes am Link beitreten müssen.

Effizienzsteigerung durch Multicast

Um die Belastung auf dem Link gering zu halten, sollten Pakete für eine Multicast Group zwar an alle beigetretenen Interfaces zugestellt werden, unbeteiligte Interfaces aber außen vor gelassen werden. Da sich 128 Bits lange Multicast Addresses nicht ohne Verlust auf gängige Link-layer Addresses abbilden lassen, muss man hier Einschränkungen hinnehmen. Je nach verwendeter Link-Technologie und Intelligenz der beteiligten Link-layer-Geräte (Beispielsweise Switches), ist der Overhead kleiner oder größer. Im ungünstigsten Fall sinkt die Effizienz auf das Niveau von Broadcast.

Multicast auf Ethernet

Die Umsetzung von Multicast Addresses auf Link-layer Addresses an Ethernet-Links werden wir wegen seiner praktischen Relevanz genauer untersuchen. Das Verfahren ist auch in RFC 2464 [Cra98] beschrieben.

Neighbor Solicitation mitschneiden

Zunächst fangen wir mit Wireshark wieder eine Neighbor Solicitation auf. Den Vorgang starten wir aber erst, wenn der Neighbor Cache von *fuzzball* keinen Eintrag mehr für *lynx* aufweist. Dann senden einen Echo Request von *fuzzball* an *lynx*, um eine Neighbor Solicitation zu erzwingen:

```
user@fuzzball:~$ ping6 -c 3 fe80::200:ff:fe60:d1e%eth1
PING fe80::200:ff:fe60:d1e%eth1 (fe80::200:ff:fe60:d1e) ⤸
     56 data bytes
64 bytes from fe80::200:ff:fe60:d1e: icmp_seq=1 ttl=64 ⤸
     time=3.85 ms
✂ 3 packets transmitted, 3 received, 0% packet loss, time ⤸
     2007ms
```

Das Ergebnis sollte der Abbildung 4.18 ähnlich sehen.

Abbildung 4.18 Neighbor Solicitation mittels Link-layer-Multicast

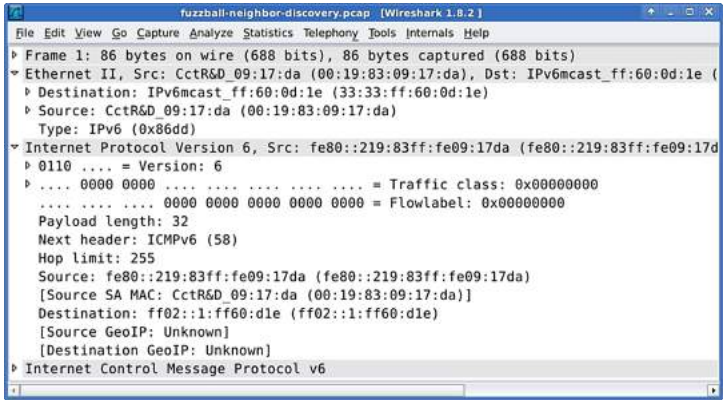
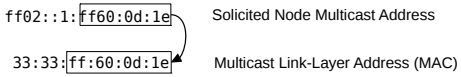


Abbildung 4.19 Link-layer Multicast Address



Solicited Node Multicast Address

In Wireshark schauen wir uns den Ethernet-Header und den IPv6-Header der Neighbor Solicitation an. Das Feld Destination im Ethernet-Header hat den Wert `33:33:ff:60:0d:1e`. Vergleichen wir den Wert mit der Zieladresse `ff02::1:ff60:d1e` im IPv6-Header, fallen Gemeinsamkeiten auf. Offensichtlich wird die Link-layer Multicast Address aus der IPv6 Multicast Address abgeleitet. Abbildung 4.19 verdeutlicht das Verfahren.

In diesem Fall sind die letzten drei Bytes der Link-layer Multicast Address identisch mit denen der Link-layer Address des Interfaces. Zur Erinnerung: Die Link-layer Address hatte uns der Node in einem Neighbor Advertisement mitgeteilt (siehe Abbildung 4.8 in Abschnitt 4.3 *Neighbor Cache*). Ein Switch müsste in diesem Fall den Frame einfach auf allen Ports aussenden, deren zugeordnete Link-layer Addresses auf die letzten drei Bytes der Link-layer Multicast Address enden. Viele werden das nicht sein. Ein so simples wie effizientes Verfahren.

Multicast und Privacy Extensions

Problematischer wird es, wenn die Clients Privacy Extensions nutzen. Dann weisen die Interface Identifier keine Gemeinsamkeiten mit der Link-layer Address mehr auf. Trotzdem bilden die Interface Identifier die Grundlage für entsprechen-

de Solicited Node Multicast Addresses. Aus diesen wiederum wird die Link-layer Multicast Address abgeleitet. Einem Switch, und sei er auch noch so schlau konfiguriert, bieten sich nun keine Anhaltspunkte mehr, auf welchen Ports der Frame erwünscht sein könnte. Ihm bleibt nur eine Möglichkeit übrig: Er sendet den Frame auf allen Ports hinaus. Um diesem Effizienzverlust zu begegnen sind die Switch-Hersteller angehalten, MLDv2-Pakete auszuwerten. Indem sich die Switche merken, an welchem Port Nodes zu einer Multicast Group beigetreten sind, können sie den Overhead signifikant senken. Dass sich dies spürbar auf die Herstellungskosten, und damit auch auf den Verkaufspreis auswirkt, dürfte auf der Hand liegen.

Sonderfall

Einen Sonderfall gibt es allerdings, beschrieben in RFC 6085 [GTTD11]: Wenn der Absender weiß, dass nur ein einziges Interface in einer Multicast Group Mitglied ist, und ihm darüber hinaus die Link-layer Unicast Address des Interfaces bekannt ist, dann darf er direkt an diese adressieren. Einem Switch wird so unter Umständen das mehrfache Aussenden eines Frames erspart.

5 | Der Link

Im Moment ist der Link *internal* noch relativ unbelebt. Die Nodes haben gelegentlich über Link-local Addresses miteinander kommuniziert, dabei die gegenseitige Erreichbarkeit nachgewiesen und ihre Neighbor Caches gepflegt. Was den Hosts fehlt, ist der Zugang zur großen weiten IPv6-Welt. Da könnte *fuzzball* weiterhelfen, er hat dank des Tunnels bereits Zugang zu ihr. Im Folgenden werden wir daher den Router *fuzzball* bestimmungsgemäß einsetzen, die Hosts mit gültigen, weltweit eindeutigen Adressen versorgen sowie ihre Konnektivität überprüfen.

5.1 | Ein Präfix für den Link

Automatische Konfiguration

Der Link *internal* soll ein Global-Unicast-Präfix erhalten um am IPv6-Internet teilnehmen zu können. Eine manuelle Konfiguration der Hosts am Link möchten wir möglichst vermeiden. Nach einer automatisierten Konfiguration sollen die Hosts mit

- einer oder mehreren IPv6-Adressen,
- der Adresse des zuständigen Routers und
- der Adresse eines *Resolving Nameservers*

ausgestattet sein.

Resolving Nameserver

Der Resolving Nameserver wäre, technisch gesehen, für das Herstellen der Konnektivität nicht erforderlich. In der Praxis erweisen sich Hosts ohne funktionierende Namensauflösung

aber als mühsam bedienbar. Für die tägliche Arbeit mit dem Internet sind sie schlicht nicht geeignet.

Aufgabenteilung

Welche Teile der automatischen Konfiguration der Router erledigt, und wo die Hosts selbst tätig werden müssen, werden wir gleich erfahren. Soviel vorweg: Die Konfigurationen werden nicht als mundfertige Häppchen serviert, wie es bei IPv4 in Verbindung mit DHCP üblich ist. Einer erfolgreichen Konfiguration geht immer ein Zusammenspiel von Router und Host voraus.

Präfix aktivieren

Ein 64 Bits langes Präfix haben wir von SixXS als kostenlose Beigabe zum Tunnel erhalten. Das Präfix werden wir für die Adressierung der Nodes auf dem internen Link verwenden. Dazu stellen wir sicher, dass das Präfix bei SixXS aktiviert ist. Im Home-Bereich der SixXS-Website befindet sich eine Tabelle mit Präfixen, die dort *Subnets* genannt werden (siehe auch Abbildung 3.11 auf Seite 53). In der Spalte *State* sollte für das zu unserem Tunnel gehörende Präfix **Enabled** stehen. In der Spalte *Subnet Präfix* finden wir jenes Präfix, welches über unseren Tunnel geroutet wird. In den Beispielen im Workshop lautet es `2a01:198:200:8a23::/64`. Wir notieren uns das eigene Präfix für die spätere Verwendung.

Router Advertisement

Unter IPv6 gehört es zum guten Ton, dass sich ein Router am Link bekannt macht. Dabei liefert er auch Informationen zu den verwendeten Präfixen mit, insbesondere zu den Präfixen für die er selbst zuständig ist. Eine solche Information wird *Router Advertisement* genannt. Router Advertisements sind Teil von NDP und werden, wie von NDP gewohnt, über ICMPv6 transportiert. Wir werden *fuzzball* dazu bewegen, solche Router Advertisements am Link *internal* auszusenden.

Router Advertisement Daemon

Dazu werden wir den *Router Advertisement Daemon*, kurz *Radvd*, verwenden. Die Installation gelingt, wie gewohnt, über den Paketmanager:

Datei: /etc/radvd.conf

```
1 interface eth1 {
2         AdvSendAdvert on;
3         MinRtrAdvInterval 15;
4         prefix 2a01:198:200:8a23::/64 { };
5 };
```

Abbildung 5.1 Minimale Konfiguration von Radvd

```
root@fuzzball:~# apt-get install radvd
✂ The following NEW packages will be installed:
  radvd
0 upgraded, 1 newly installed, 0 to remove and 0 not ⤸
    upgraded.
✂ Starting radvd:
* /etc/radvd.conf does not exist or is empty.
* See /usr/share/doc/radvd/README.Debian
* radvd will *not* be started.
```

Radvd wird mit einer ausführlichen Dokumentation ausgeliefert. Die Dokumentation lässt sich mit den Kommandos `man radvd` und `man radvd.conf` anzeigen.

Erstkonfiguration von Radvd

Aufgrund fehlender Konfigurationsdatei verweigert Radvd noch die Aufnahme der Arbeit. Die Konfiguration reichen wir sogleich nach. Dazu bearbeiten wir die Datei `/etc/radvd.conf` gemäß Abbildung 5.1.

Für jedes Interface auf dem Radvd Router Advertisements versenden soll, enthält die Konfigurationsdatei einen *interface*-Block. Das eigentliche Senden erlaubt oder verbietet der Parameter *AdvSendAdvert*. Mit *MinRtrAdvInterval* legen wir die Zeit in Sekunden fest, die zwischen zwei periodisch versandten Router Advertisements vergehen soll. 15 Sekunden ist ein guter Wert für einen Link dieser Größe. Anstelle von `2a01:198:200:8a23::/64` setzen Sie bitte das Präfix ein, dass Sie sich vorhin von der SixXS-Seite notiert haben.

Interface konfigurieren

Bevor wir Radvd neu starten, erhält das Interface *eth1* noch eine Adresse aus dem Präfix das verteilt werden soll:

```
root@fuzzball:~# ip addr add 2a01:198:200:8a23::1/64 dev ⤸
    eth1
```

Abbildung 5.2 Permanente Konfiguration von *eth1*

Datei: `/etc/network/interfaces`

```
1  # Loopback Interface
2  auto lo
3  iface lo inet loopback
4
5  # Interface an VirtualBox auf dem Wirtsystem
6  allow-hotplug eth0
7  iface eth0 inet dhcp
8
9  # Interface am Link "internal"
10 auto eth1
11 iface eth1 inet6 static
12         address 2a01:198:200:8a23::1
13         netmask 64
```

Der wichtigste Effekt des letzten Schritts ist eine Veränderung in der Routingtabelle von *fuzzball*:

```
user@fuzzball:~$ ip -6 route show
2a01:198:200:a23::/64 dev sixxs  proto kernel  metric ⤸
     256  mtu 1280 advmss 1220 hoplimit 4294967295
2a01:198:200:8a23::/64 dev eth1  proto kernel  metric ⤸
     256  mtu 1500 advmss 1440 hoplimit 4294967295
✂ default via 2a01:198:200:a23::1 dev sixxs  metric 1024  ⤸
     mtu 1280 advmss 1220 hoplimit 4294967295
```

Ohne den Eintrag für *eth1* liefe *fuzzball* Gefahr, ein Präfix auf Interface *eth1* zu verteilen, zu dem er selbst keine Route kennt. Für einen Router eine sehr ungünstige Situation, denn diese Pakete würde er dann schlicht verwerfen.

Damit das Interface auch nach dem nächsten Neustart wieder so konfiguriert wird, aktualisieren wir die Datei `/etc/network/interfaces` auf *fuzzball* gemäß Abbildung 5.2.

Die Konfiguration des Interfaces wird ab jetzt statisch erfolgen. Die Direktiven *up* und *down* haben nun ausgedient und werden durch *address* und *netmask* ersetzt.

Forwarding aktivieren

Ein Router ist, wir erinnern uns an den Abschnitt 2.4 *Begriffsdefinitionen*, ein Node der Pakete weiterleitet. Als Fachbegriff dafür hat sich das englische *forwarding* durchgesetzt. Das Forwarding müssen wir *fuzzball* explizit erlauben, denn aus Sicherheitsgründen ist es standardmäßig deaktiviert. Dazu editieren wir die Datei `/etc/sysctl.conf` wie in Abbildung 5.3 gezeigt.

Datei: `/etc/sysctl.conf` (Auszug)

```
1 # Uncommment the next line to enable packet forwarding ⤸
      for IPv6
2 # Enabling this option disables Stateless Address ⤸
      Autoconfiguration
3 # based on Router Advertisements for this host
4 net.ipv6.conf.all.forwarding=1
```

Abbildung 5.3 IPv6-Forwarding permanent aktivieren

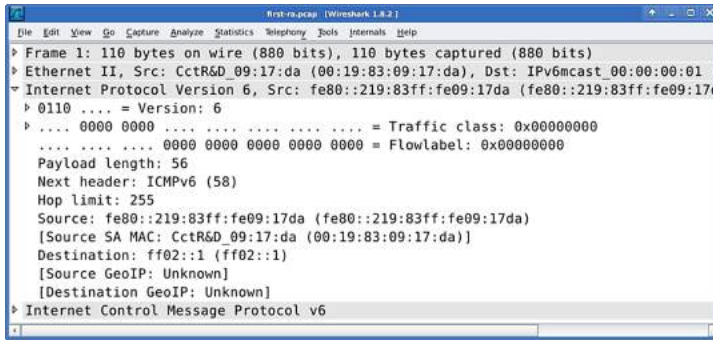


Abbildung 5.4 IPv6-Header eines Router Advertisements

Die Änderungen werden erst nach dem nächsten Neustart aktiv. Alternativ können wir die Prozedur abkürzen und die Änderungen sofort wirksam werden lassen:

```
root@fuzzball:~# sysctl -p
net.ipv6.conf.all.forwarding = 1
```

Jetzt kann *fuzzball* anfangen Router Advertisements zu versenden. Wir starten den zuständigen Daemon:

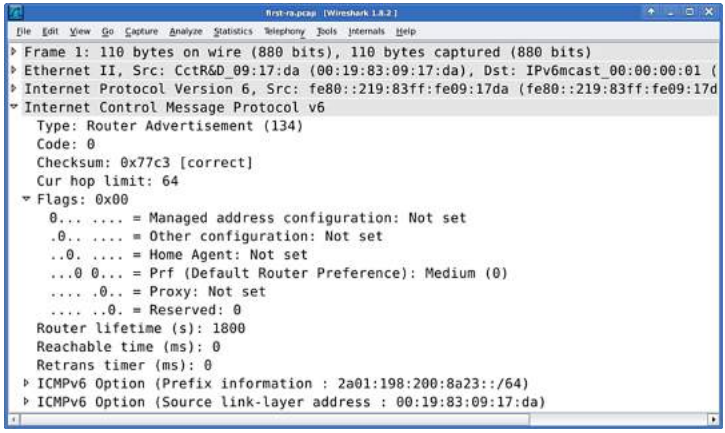
```
root@fuzzball:~# service radvd start
Starting radvd: radvd.
```

Nun sollte Radvd periodische Router Advertisements aussenden.

Router Advertisement mitschneiden

Natürlich werden wir auch den Beweis antreten und ein solches Router Advertisement auf dem Link entdecken. Wir starten Wireshark auf *lynx* und lassen ihn auf *eth0* lauschen. Spätestens nach 15 Sekunden sollte das erste Router Advertisement aufgefangen werden. Das werden wir uns nun genauer anschauen. Es sollte den Abbildungen 5.4 und 5.5 recht nahe kommen.

Abbildung 5.5 Router Advertisement nach Minimalkonfiguration



IPv6-Header des Router Advertisements

Betrachten wir zunächst den IPv6-Header des Paketes (Abbildung 5.4). Das Feld Next Header verweist auf ICMPv6, dem Transportprotokoll von NDP. Folgerichtig ist das Hop Limit auf 255 gesetzt. Die Quelladresse ist die Link-local Address von *eth1*, was von RFC 2461 [NNS98] so vorgeschrieben wird. Die Zieladresse für periodisch versandte Router Advertisements ist die All Nodes Multicast Address `ff02::1`. Ihr Multicast Scope ist, das haben wir bereits gelernt, an der vierten hexadezimalen Stelle verzeichnet. Hier handelt es sich um den Scope Link, da wir an vierter Stelle eine 2 finden. Von den Flags, an dritter hexadezimaler Stelle zu finden, ist keines gesetzt. Denn nur wenn kein Flag gesetzt ist, kann an die dritte hexadezimale Stelle 0 lauten. Somit ist auch das Transient-Flag nicht gesetzt. Daher können wir uns sicher sein, dass es sich bei dieser Multicast Address um eine Well-known Multicast Address handelt. Tatsächlich finden wir sie in Tabelle 4.5 auf Seite 103 wieder. Das Wissen um den Aufbau von Multicast Addresses hat uns bereits ein erstes Mal weitergeholfen.

Router Advertisement im Detail

In der ICMPv6-Nachricht (Abbildung 5.5) sehen wir an erste Stelle die Typnummer 134 für Router Advertisements. Nach Code und Checksum folgt ein Feld namens *Cur Hop Limit*, es hat in unserem Beispiel den Wert 64. Das Cur Hop Limit gibt an, welches Hop Limit ein Host, der seinen Stack gemäß den Angaben dieses Router Advertisements kon-

figuriert, nutzen soll. Hat das Feld den Wert 0, dann gibt es keine Vorgabe durch den Router, der Host entscheidet dann selbst welches Hop Limit er für IPv6-Pakete verwendet.

Router Advertisement Flags

Es folgen Flags, von denen hier keines gesetzt ist. Möglich wären folgende Flags:

Managed
: Das *Managed Flag* zeigt einem interessierten Host an, dass er auf anderen Wegen Adressen erhalten kann. Andere Wege meint zum Beispiel DHCPv6 und wird auch als *stateful configuration* bezeichnet. Nicht zu verwechseln mit der im Deutschen ähnlichen klingenden *statischen Konfiguration*.

Other Managed
: Mit diesem Flag informiert der Router darüber, dass weitere Konfigurationsdaten über einen anderen Weg bezogen werden können. Auch hier ist in der Regel DHCPv6 gemeint, welches beispielsweise Nameserver-Adressen zur Verfügung stellen könnte.

Home Agent
: Ein gesetztes Flag zeigt an, dass dieser Router als *Home Agent* für Mobile IPv6 genutzt werden kann.

Router Preference
: Eigentlich handelt es sich bei der *Router Preference* nicht um ein Flag im klassischen Sinne. Es werden zwei Bits im Flag-Feld genutzt um die Priorität des Routers zu codieren. Es gibt die Werte low, medium und high. Sollten mehrere Default-Router am Link präsent sein, wird die Priorität auf den Hosts unter Berücksichtigung dieses Flags festgelegt.

Proxy
: Dieses Flag wurde im experimentellen RFC 4389 [TTP06] vorgeschlagen. Es deutet auf das Vorhandensein eines NDP-Proxys hin. Für uns ist das Flag nicht relevant und wir behandeln es im Folgenden nicht weiter.

Die Flags können, geschickt kombiniert, sehr unterschiedliche Situationen abdecken. So wäre es zum Beispiel möglich, den

Hosts zwar die eigenständige Adresskonfiguration zu erlauben, ihnen alle zusätzlichen Informationen jedoch per DHCPv6 zuzustellen.

Router Lifetime

Im Router Advertisement folgt als nächstes die *Router Lifetime* in Sekunden. Sie gibt an, wie lange dieser Router als Default-Router verwendet werden darf. Der Wert 0 bedeutet, dass der Router nicht als Default-Router verwendet werden darf. In unserem Fall bedeutet das, dass der Router nach 1800 Sekunden (einer halben Stunde) ungültig werden würde. Mit jedem neuen Router Advertisement kann dieser Wert überschrieben werden. Da wir alle 15 Sekunden ein Router Advertisement aussenden, sollten wir vorerst nicht Gefahr laufen, den Default-Router zu verlieren.

Reachable Time und Retrans Timer

Die Felder *Reachable Time* und *Retrans Timer* dienen den Stacks der Hosts als Anhaltspunkte für ihre eigene Konfiguration. Die Reachable Time legt fest, wie viele Millisekunden ein Nachbar auf dem Link als erreichbar gilt, nachdem eine Bestätigung der Erreichbarkeit eingetroffen ist.[1] Der Retrans Timer legt die Wartezeit zwischen zwei Neighbor Solicitations in Millisekunden fest. Die Werte dieser Felder haben also direkten Einfluss auf die Neighbor Caches der Hosts. Wenn die Felder den Wert `0` enthalten, macht der Router keine Vorgaben und die Hosts können ihre eigene Konfiguration verwenden.

Präfix-Informationen

Unser Router Advertisement liefert zwei ICMPv6-Optionen mit. Die Option *Source Link-layer Address* kennen wir schon aus anderen NDP-Paketen. Die *Prefix Information Option* aus Abbildung 5.6 hingegen ist uns neu.

Wie jede ICMPv6-Option hat sie eine Typnummer und eine Längenangabe. Daran schließt sich das Feld *Prefix Length* an, es gibt an wie viele Bits des Präfixes von Router vorgegeben werden. In diesem Fall sind es 64 Bits, so dass den Hosts

[1] Als Bestätigung der Erreichbarkeit können alle Pakete dienen, die darauf schließen lassen, das der Nachbar erreichbar ist. Zum Beispiel können das eintreffende Pakete einer bestehenden TCP-Verbindung sein. Es muss sich nicht zwingend um NDP handeln.

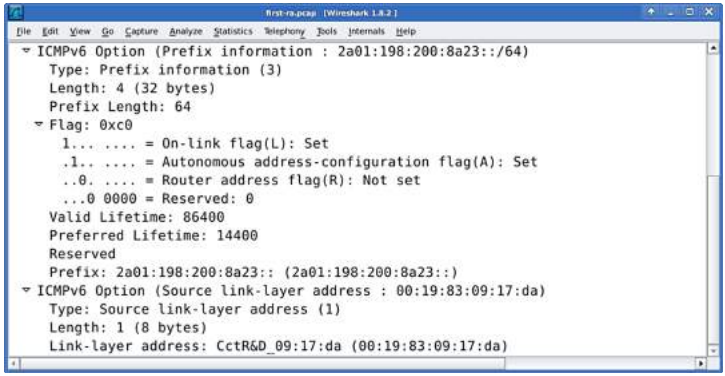


Abbildung 5.6
Prefix Information Option

weitere 64 Bits für ihre Interface Identifier zur Verfügung stehen.

Präfix-Flags

Jedes Präfix hat ein eigenes Feld für Flags. Die Bedeutung der Flags ist wie folgt:

On-Link
: Ist das Flag gesetzt, zeigt der Router damit an, dass dieses Präfix sich auf dem Link befindet. Pakete an Adressen innerhalb des Präfixes können direkt gesendet werden, und brauchen nicht über den Router gehen. Wenn das Flag nicht gesetzt wurde, bedeutet das lediglich, dass der Router keine Aussage treffen konnte.

Autonomous Address-Configuration
: Das Flag wird gesetzt wenn das Präfix den Hosts für die Stateless Address Autoconfiguration zur Verfügung steht.

Router Address
: Wenn das Flag gesetzt ist, befindet sich im Präfix-Feld anstelle eines Präfixes die Adresse des Interfaces welches für Mobile IPv6 zuständig ist. Dieses Flag nutzen wir im Workshop nicht.

Valid Lifetime und Preferred Lifetime

In den Feldern Valid Lifetime und Preferred Lifetime wird festgelegt, wie lange das Präfix gültig (valid) ist und bevorzugt (preferred) werden soll. Generierte Adressen aus dem Präfix erben diese Eigenschaften. Für neue Verbindungen werden stets bevorzugte Adressen genutzt. Adressen bestehen-

der Verbindungen können während der Valid Lifetime noch genutzt werden, dürfen aber nicht für neue Verbindungen hergenommen werden. Die Angaben erfolgen jeweils in Sekunden. Der höchstmögliche Wert steht abweichend für unbegrenzte Gültigkeit.

Das Präfix

Das letzte, und wohl wichtigste, Feld ist das Präfix selbst. Der Teil hinter der mit Prefix Length definierten Grenze wird üblicherweise mit Nullen aufgefüllt und von den Hosts ignoriert. Unser Router hat hier das Präfix `2a01:198:200:8a23::` verteilt, so wie wir es Radvd aufgetragen haben.

Angepasstes Router Advertisement

Jetzt, wo wir uns mit Router Advertisements besser auskennen, werden wir unsere Minimalkonfiguration von Radvd etwas leistungssteigern. Die Router Advertisements sollen weiterhin in Abständen von mindestens 15 Sekunden ausgesendet werden, es sollen aber nie mehr als 60 Sekunden zwischen ihnen liegen. Das Hop Limit und die Flags erhalten Standardwerte, diese sollen zur Übung dennoch in der Konfigurationsdatei vermerkt werden. Der Router soll mit mittlerer Priorität als Default-Router angepriesen werden.

Da der Tunnel in der Standardkonfiguration eine MTU von 1280 aufweist, propagieren wir diese MTU auf dem Link. Die Link MTU wird später als weitere ICMPv6-Option im Router Advertisement auftauchen.

Das Präfix soll mit gesetzten Flags für On-Link und Autonomous Address Autoconfiguration verteilt werden. Generierte Adressen sollen eine Stunde lang gültig sein und für eine halbe Stunde bevorzugt werden.

Natürlich darf auch die Source Link-layer Address nicht fehlen.

Radvd konfigurieren

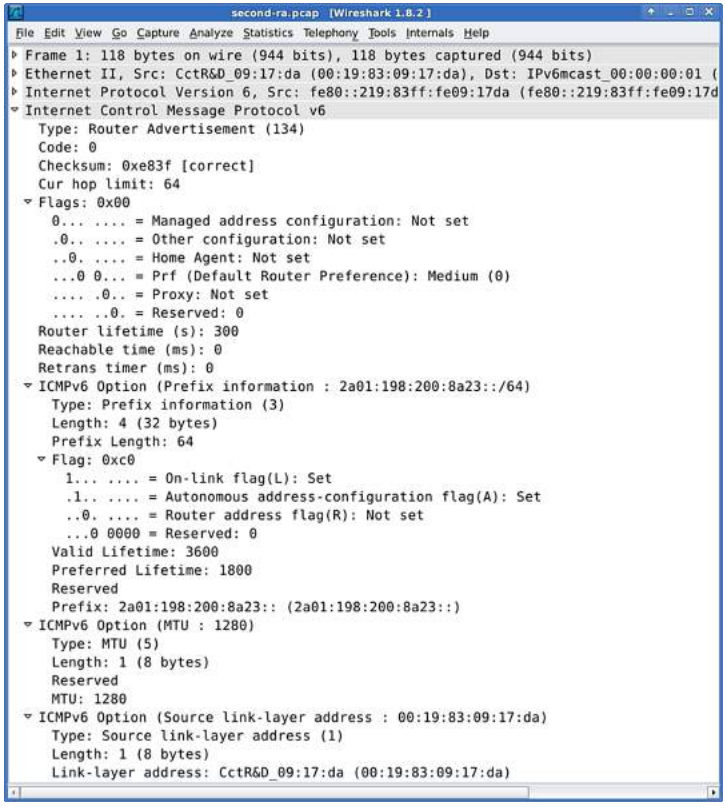
Versuchen Sie mithilfe des Kommandos `man radvd.conf` das eben beschriebene Router Advertisement zu konfigurieren. Die Lösung finden Sie in Abbildung 5.7.

Datei: /etc/radvd.conf

```
interface eth1 {
        AdvSendAdvert on;
        MinRtrAdvInterval 15;
        MaxRtrAdvInterval 60;

        AdvCurHopLimit 64;

        AdvManagedFlag off;
        AdvOtherConfigFlag off;
        AdvMobRtrSupportFlag off;
        AdvDefaultPreference medium;

        AdvDefaultLifetime 300;
        AdvReachableTime 0;
        AdvRetransTimer 0;

        AdvLinkMTU 1280;

        prefix 2a01:198:200:8a23::/64 {
                AdvOnLink on;
                AdvAutonomous on;
                AdvRouterAddr off;

                AdvValidLifetime 3600;
                AdvPreferredLifetime 1800;
        };

        AdvSourceLLAddress on;
};
```

Abbildung 5.7 Angepasste Konfiguration von Radvd

Abbildung 5.8 Angepasstes Router Advertisement



Zum Anwenden der geänderten Konfiguration starten wir Radvd neu:

```
root@fuzzball:~# service radvd restart
Stopping radvd: radvd.
Starting radvd: radvd.
```

Das neue Router Advertisement fangen wir ebenfalls mit Wireshark auf *lynx* auf. Es ist zur Kontrolle in Abbildung 5.8 zu sehen.

5.2 | Stateless Address Autoconfiguration

Zweck

Die *Stateless Address Autoconfiguration*, kurz SLAAC, dient der automatischen Konfiguration von Adressen und Routen der Hosts am Link. Damit reduziert IPv6 als Protokoll die

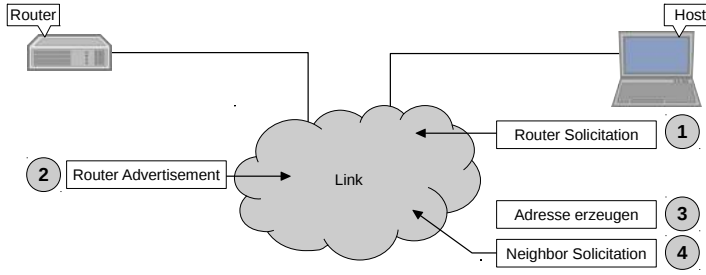


Abbildung 5.9 Prinzip von SLAAC

Abhängigkeit von dritten Komponenten zur Organisation des Links. Die Nutzung von Stateless Address Autoconfiguration erfordert keine manuelle Konfiguration der Hosts und nur sehr wenige Konfigurationsschritte auf dem Router. Damit einher geht der Verlust einer strengen Zuordnung von Adressen zu bestimmten Hosts. In Umgebungen wo die Zuordnung von Adressen zu Hosts zentral gesteuert werden soll, ist dieser Ansatz nicht ausreichend. Dort würde man auf DHCPv6 zurückgreifen. Aber auch ein simultaner Betrieb von DHCPv6 und Stateless Address Autoconfiguration wäre denkbar.

Autoconfiguration

SLAAC ist Bestandteil der *Autoconfiguration*, die drei wesentliche Aufgaben hat:

- Generieren einer Link-local Address
- Durchführen der Stateless Address Autoconfiguration
- Sicherstellen der Eindeutigkeit der generierten Adressen (Duplicate Address Detection)

Prinzipieller Ablauf

Der prinzipielle Ablauf der Stateless Address Autoconfiguration ist in Abbildung 5.9 zu sehen.

Den ersten Schritt macht der Host, indem er mittels *Router Solicitation* nach einem Router Advertisement fragt. Alternativ könnte er auch ein periodisches Router Advertisement abwarten, diese Geduld beobachtet man aber eher selten.

Der Router verschickt das angeforderte Router Advertisement, welches alle konfigurationsrelevanten Daten enthält.[2]

[2] Wir gehen der Einfachheit halber von nur einem Router aus.

Daraufhin führt der Host die Konfiguration des Interfaces durch und prüft die Eindeutigkeit der selbst erzeugten Adressen. Erst wenn diese Eindeutigkeit angenommen werden kann, ist die Konfiguration des Interfaces vollständig und gilt als beendet.

Duplicate Address Detection

Hinter der Duplicate Address Detection verbergen sich eigentlich mehrere Neighbor Solicitations. Wenn ein Node feststellen möchte, ob eine Adresse schon von einem anderen Node genutzt wird, dann versucht er die zugehörige Link-layer Address aufzulösen. Bleibt eine Antwort aus, benutzt offensichtlich kein anderer Node auf dem Link die überprüfte Adresse. Um Fehlschlüsse aufgrund von Paketverlusten zu vermeiden, sollen mehrere Neighbor Solicitations verschickt werden.

Ab wann eine Adresse als eindeutig gilt, hängt von den Parametern der jeweiligen Implementierung ab. Jede Adresse hat anfangs den Status *tentative* (probeweise). Erst wenn die Duplicate Address Detection vollständig durchlaufen wurde, und keine Anzeichen darauf schließen lassen, dass die Adresse bereits in Benutzung ist, wird die Adresse *valid* (gültig).

Auto-configuration mitschneiden

Wir werden versuchen eine komplette Autoconfiguration von *lynx* mit Wireshark aufzufangen. Vom Hochfahren des Interfaces bis zu seiner endgültigen Konfiguration.

Dazu öffnen wir ein root-Terminal auf *lynx* und fahren das Interface *eth0* herunter:

```
root@lynx:~# ip link set down dev eth0
```

Nun starten wir Wireshark und lassen ihn auf dem Pseudo-Interface *any* lauschen.

Danach fahren wir *eth0* wieder hoch:

```
root@lynx:~# ip link set up dev eth0
```

In Wireshark können wir bereits Aktivität beobachten. Wir warten den Abschluss der Konfiguration ab, sie ist erfolgreich verlaufen wenn wir Adressen mit den Parametern *scope global* und *dynamic* sehen:

```
user@lynx:~$ ip addr show dev eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 ↩
     qdisc pfifo_fast state UP qlen 1000
    link/ether 00:00:00:60:0d:1e brd ff:ff:ff:ff:ff:ff
    inet6 2a01:198:200:8a23:200:ff:fe60:d1e/64 scope ↩
         global dynamic
       valid_lft 3578sec preferred_lft 1778sec
    inet6 fe80::200:ff:fe60:d1e/64 scope link
       valid_lft forever preferred_lft forever
```

Wireshark kann jetzt den Mitschnitt beenden. Von den vielen Paketen die wir mitgeschnitten haben sind nicht alle von Interesse.

Multicast DNS

Insbesondere die Pakete vom Typ *Multicast DNS* (mDNS) werden wir an dieser Stelle ignorieren. Multicast DNS erlaubt die Auflösung von Namen der Domain `.local` zu Link-local Addresses. Leider belastet es dazu den Link ungefragt mit allerlei Paketen. Da wir im Workshop keine lokale Namensauflösung auf Multicast-Basis nutzen, kümmern wir uns nicht weiter um dieses Protokoll. Mehr zu Multicast DNS und seinem Nutzen für kleine Netze findet sich auf der gemeinsamen Website der Beteiligten Interessensgruppen.[3]

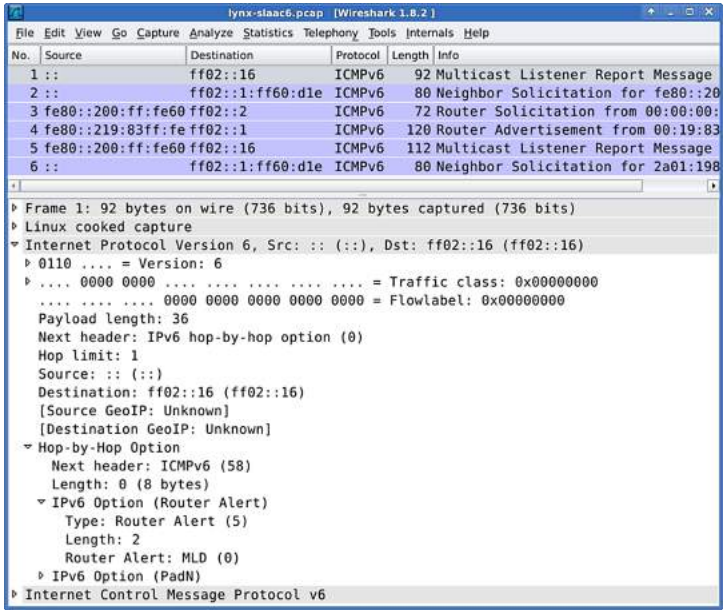
Multicast Listener Report (Solicited Node)

Das erste interessante Paket im Mitschnitt ist ein Multicast Listener Report und wurde von *lynx* verschickt. Der IPv6-Header ist in Abbildung 5.10 zu sehen.

Als Quelladresse hat *lynx* die Unspecified Address gewählt. Das heißt, zum Zeitpunkt des Versendens stand keine passende, gültige Adresse zur Verfügung. Die Zieladresse ist die Multicast Address für alle MLDv2-fähigen Router. Zu finden auch in der Tabelle 4.5 in Abschnitt 4.5 *Multicast*. Uns fällt das für MLDv2 Messages typische Hop Limit von 1 auf. Bemerkenswert ist auch der Einsatz des Hop-by-Hop Options Extension Headers. Neben der Padding Option, welche den

[3] http://www.multicastdns.org/

Abbildung 5.10 SLAAC Paket 1: IPv6-Header



Extension Header auf eine einheitliche Länge auffüllt, ist auch eine Router Alert Option vorhanden. Sie informiert Multicast-fähige Router darüber, dass sich eine MLDv2 Message im Paket befindet. Interessierte Router werten die Nachricht dann aus und ziehen daraus Schlüsse für ihr Multicast Routing. Die MLDv2 Message kann in Abbildung 5.11 eingesehen werden.

Genaugenommen handelt sich um eine MLDv2 Message der Art Changed to Exclude. Wir haben in Abschnitt 4.5 *Multicast* bereits besprochen wie dieser Typ zu interpretieren ist. Hier wird die Multicast Address `ff02::1:ff60:d1e` für alle potentiellen Multicast-Quellen freigegeben. Die Nachricht entspricht dem Beitritt zur Multicast Group `ff02::1:ff60:d1e`. Es handelt sich dabei um die Solicited Node Multicast Address von Interface *eth0* auf *lynx*.

Gruppenbeitritt

Zu diesem Zeitpunkt hat *lynx* also bereits einen Interface Identifier erzeugt. Der Beitritt zur entsprechenden Multicast Group gewährt ihm Zugang zu den Paketen dieser Gruppe. So hat er die Chance, frühzeitig zu erfahren, ob sein Inter-

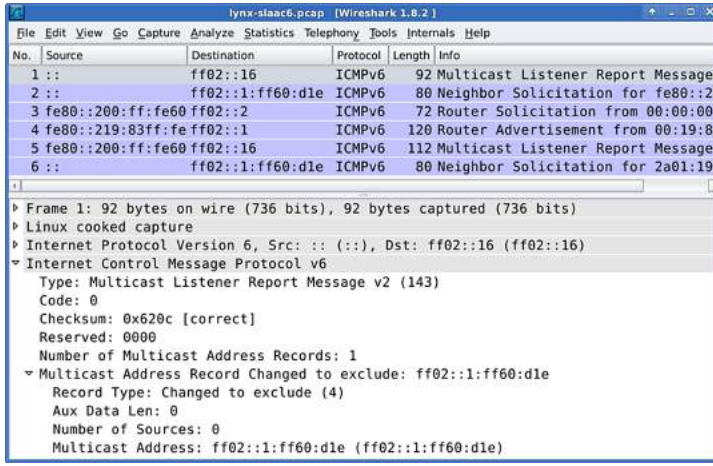


Abbildung 5.11 SLAAC Paket 1: Multicast Listener Report

face Identifier schon verwendet wird. Würde ein anderer Node seinen Interface Identifier bereits verwenden, so wäre dieser Node ebenfalls Mitglied der Multicast Group. Eine doppelt vorkommende Adresse würde dadurch schneller auffallen.

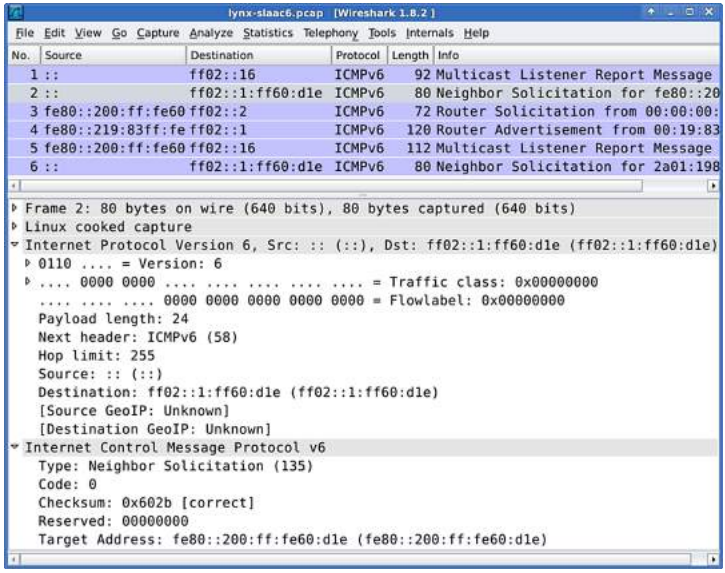
Gruppen mit mehreren Mitgliedern

Es wäre allerdings auch möglich, dass ein anderer Node eine ähnliche Adresse verwendet. Beispielsweise eine Adresse bei der sich die letzten 24 Bits gleichen. Beide Nodes wären nun in derselben Gruppe, jene mit der gemeinsamen Solicited Node Multicast Address. Beide Nodes würden auch Pakete empfangen, die nicht für sie bestimmt wären, die aufgrund der Ähnlichkeit der Adresse aber an die gemeinsame Gruppe geschickt wurden. Jeder Node muss deshalb prüfen, ob ein Paket, welches an die Gruppe adressiert wurde, auch wirklich für ihn von Belang ist. Auch *lynx* könnte Pakete empfangen, nach der Prüfung des Inhaltes aber feststellen, dass der eigene Interface Identifier davon nicht betroffen ist.

Duplicate Address Detection (Link-local)

Möchte *lynx* nun feststellen, ob die von ihm gewählte Adresse nicht nur vielleicht eindeutig ist, dann ist eine Duplicate Address Detection erfolgsversprechender. Wenn sie fehlschlägt, dann ist die von *lynx* gewählte Adresse *sehr wahrscheinlich*

Abbildung 5.12 SLAAC Paket 2: Neighbor Solicitation



auf dem Link noch nicht vergeben.[4] Dazu sendet *lynx* eine Neighbor Solicitation für die selbst erzeugte Adresse aus (siehe Abbildung 5.12).

Als Quelladresse wählt er wieder die Unspecified Address, da die Link-local Address noch nicht als eindeutig gilt. Die Neighbor Solicitation geht an die Solicited Node Multicast Address der zu überprüfenden Link-local Address. Im Feld Target Address taucht die gewünschte Adresse `fe80::200:ff:fe60:d1e` schließlich auf.

Das Ausbleiben eines Neighbor Advertisements wertet der Node als Anzeichen für die Eindeutigkeit seiner Adresse auf dem Link. Sie wird dann dem Interface zugewiesen und gilt fortan als *valid*.

Router Solicitation

Nachdem *lynx* nun eine gültige Link-local Address hat, versucht er auch eine gültige Adresse für den Global Scope zu erhalten. Dazu lässt er sich von jedem Router am Link ein Router

[4] Eine endgültige Gewissheit ist mit der Duplicate Address Detection nicht zu erreichen. Im ungünstigsten Fall gehen genau jene Pakete verloren, die auf eine doppelte Adresse hinweisen würden.

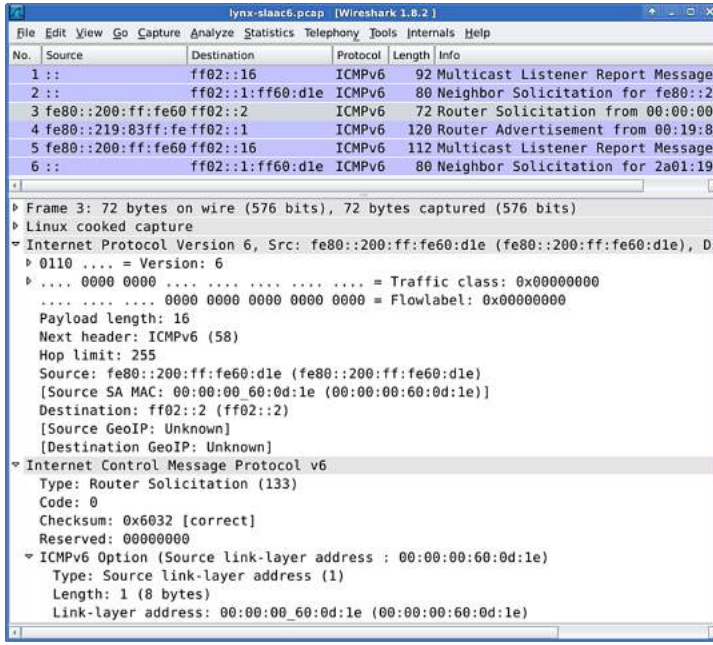


Abbildung 5.13 SLAAC Paket 3: Router Solicitation

Advertisement zukommen. Die Anforderung der Router Advertisements geschieht mit Hilfe einer Router Solicitation, die in Abbildung 5.13 zu sehen ist.

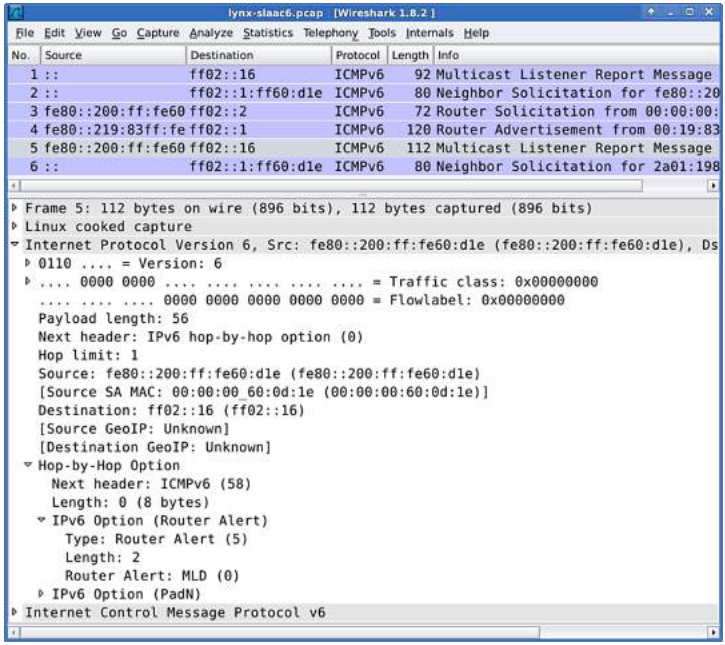
Die Nachricht wird von der Link-local Address des Hosts gesendet, hier von der Adresse `fe80::200:ff:fe60:d1e`. Als Zieladresse wird die All Routers Multicast Address `ff02::2` verwendet, die wir schon in der Tabelle 4.5 in Abschnitt 4.5 *Multicast* gesehen haben. Angehängt an die Router Solicitation ist, eine ICMPv6-Option mit der Link-layer Address des Absenders.

Router Advertisement

Alle Router am Link antworten auf die Router Solicitation mit einem Router Advertisement. Da wir nur einen Router am Link haben, nämlich *fuzzball*, erhalten wir auch nur ein Router Advertisement.

Wir werden es hier nicht genauer besprechen, denn das haben wir in Abschnitt 5.1 *Ein Präfix für den Link* schon getan. Ein auffrischender Blick in das Paket wird aber sicher nicht schaden.

Abbildung 5.14 SLAAC Paket 5: IPv6-Header



Nach dem Erhalt des Router Advertisements erzeugt *lynx* eine Global Unicast Address. Dazu verwendet er das von *fuzzball* verteilte Präfix und den bereits vorhandenen Interface Identifier.

Multicast Listener Report (Solicited Node, Multicast-DNS)

Auch für diese Adresse muss eine Duplicate Address Detection durchgeführt werden. Die beginnt wieder mit dem Beitritt zu der passenden Multicast Group. Obwohl sich die Solicited Node Multicast Address für die Global Unicast Address nicht von der für die Link-local Address unterscheidet, versendet *lynx* einen neuen Multicast Listener Report. Der wesentliche Unterschied ist die Quelladresse des Paketes, siehe auch Abbildung 5.14.

Anstatt der Unspecified Address kommt diesmal die Link-local Address zum Einsatz. Der Rest des Paketes ist in Abbildung 5.15 dargestellt.

Unverändert geblieben ist der Beitritt zur Solicited Node Multicast Group, der erneut mithilfe von Changed to Exclude erreicht wurde. Und einen weiteren Gruppenbeitritt können wir

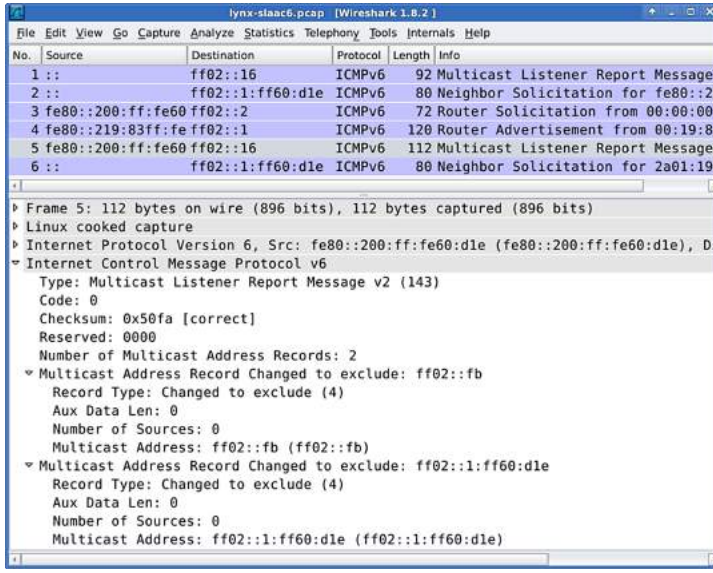


Abbildung 5.15 SLAAC Paket 5: Multicast Listener Report

im Paket entdecken, der Beitritt zur Gruppe `ff02::fb`. Dies ist die Multicast DNS Address für die Verwendung mit IPv6, und für unser Netz nicht weiter wichtig.

Duplicate Address Detection (Global Unicast)

Der letzte Schritt ist die Durchführung der Duplicate Address Detection für die Global Unicast Address. Dazu sendet *lynx* wieder eine Neighbor Solicitation, zu sehen in Abbildung 5.16, aus.

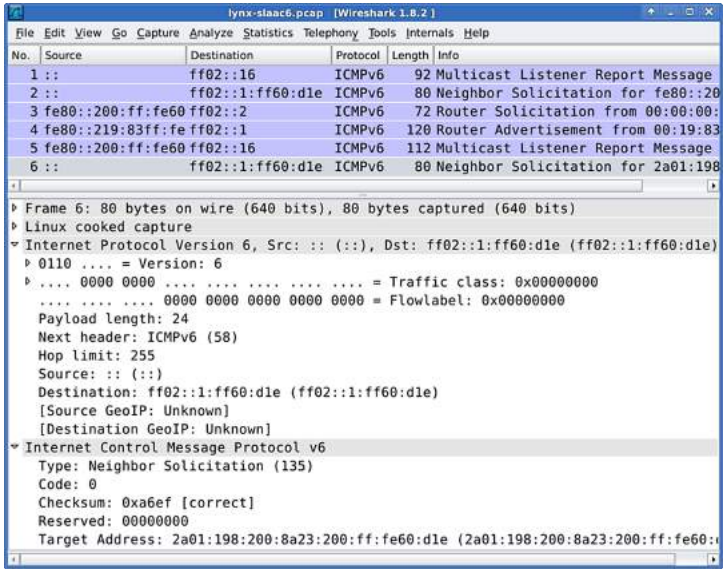
Mit dem Ausbleiben einer Antwort ist die Stateless Address Autoconfiguration abgeschlossen. Das Interface *eth0* ist nun fertig konfiguriert.

Konnektivitätstest

Einem Test der Konnektivität von *lynx* steht nun nichts mehr im Wege. Dazu verschicken wir Echo Requests von *lynx* an den Tunnelendpunkt von SixXS:

```
user@lynx:~$ ping6 -c 3 2a01:198:200:a23::1
PING 2a01:198:200:a23::1 (2a01:198:200:a23::1) 56 data ⤸
     bytes
64 bytes from 2a01:198:200:a23::1: icmp_seq=1 ttl=63 ⤸
     time=8.02 ms
✂
3 packets transmitted, 3 received, 0% packet loss, time ⤸
     2003ms
```

Abbildung 5.16 SLAAC Paket 6: Neighbor Solicitation



Da Echo Replies eintreffen, können wir davon ausgehen dass das Routing funktioniert. Den Beweis können wir auch mit `traceroute6` antreten:

```
user@lynx:~$ traceroute6 -n 2a01:198:200:a23::1
traceroute to 2a01:198:200:a23::1 (2a01:198:200:a23::1), ↩
     30 hops max, 80 byte packets
 1  2a01:198:200:8a23::1  2.204 ms  0.162 ms  0.193 ms
 2  2a01:198:200:a23::1  13.255 ms  13.412 ms  19.135 ms
```

An erster Stelle steht der nächste Hop, in unserem Fall die Adresse des Interfaces *eth1* von *fuzzball*. Schon in der zweiten Zeile ist das Ziel erreicht.

SLAAC unter Windows 8

Nun werden wir die eben erworbenen Fähigkeiten zur Analyse einer Autoconfiguration auf *felis* anwenden. Bei dieser Gelegenheit werden wir auch Unterschiede entdecken, die durch Aktivierung von Privacy Extensions auftreten. Dazu öffnen wir als Administrator ein Terminal und stellen sicher das die Privacy Extensions aktiviert sind. Nach dem Betätigen der Tastenkombination Windowstaste+X erscheint ein Menü in dem wir den Punkt *Command Prompt (Admin)* auswählen:

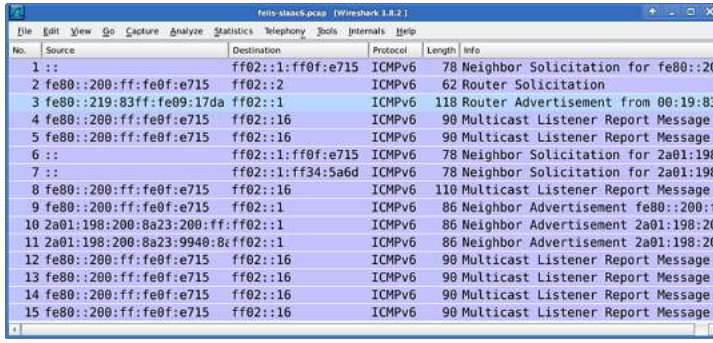


Abbildung 5.17 SLAAC unter Windows 8

```
C:\Users\user>netsh interface ipv6 set global ↵
     randomizeidentifiers=enabled
Ok.
```

Leider kommt es unter Windows 8 beim Betrieb von Wireshark manchmal zu Problemen. Der benötigte Treiber zum Mitschnitt von Daten heißt *Windows Packet Capture* (WinPcap), je nach Update-Stand von *felis* kann er funktionieren oder auch nicht. Als Lösung bietet es sich an, den Verkehr von *eth1* auf *fuzzball* mitzuschreiben, auch dort kommen die Pakete vorbei.

Sobald Wireshark bereit ist, deaktivieren wir die LAN-Verbindung auf *felis* und aktivieren sie anschließend wieder. Bei einer Beobachtung von *fuzzball* aus, können wir alternativ auch einen Neustart von *felis* durchführen. In beiden Fällen ergibt sich ein Mitschnitt, der dem aus Abbildung 5.17 ähnlich sieht.

Eigene Untersuchungen

Untersuchen Sie die einzelnen Pakete und finden Sie heraus, zu welchem Zweck jedes einzelne versendet wurde. Sie können sich dabei auf ICMPv6 beschränken und auch die Teile von MLDv2, die sich um Multicast DNS drehen, ignorieren. Erkennen Sie anhand der Informationen in den Paketen, ob diese sich auf einen zufälligen (Privacy Extensions) oder auf einen EUI-64-basierten Interface Identifier beziehen?

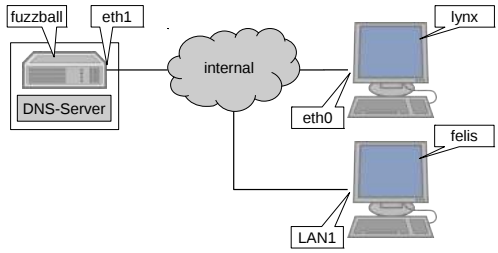


Abbildung 5.18 Interner Link mit DNS-Server

5.3 | Namensauflösung

Was den Nodes am Link jetzt noch fehlt, ist ein Nameserver für die Namensauflösung. Ein kleiner Test deckt den Missstand auf:

```
user@lynx:~$ ping6 -c 3 ping.ipv6-workshop.de
unknown host
```

Ohne die Möglichkeit Namen aufzulösen, verschafft selbst die beste Konnektivität wenig Vorteile.

Zielkonfiguration DNS

Der Router agiert bereits als zentraler Anlaufpunkt für alle Daten die den Link verlassen sollen. Es bietet sich daher an, ihm auch die Aufgabe der Namensauflösung zuzuteilen. Die Zielkonfiguration ist in Abbildung 5.18 zu sehen.

Der Router selbst, aber auch die Hosts können dann DNS-Anfragen an den installierten *Resolving DNS Server* (RDNSS) weiterleiten. Dieser kümmert sich um die Auflösung des Namens und teilt dem Anfragenden das Ergebnis mit. Der Einsatz eines Caches sorgt dafür, das zukünftige Anfragen teilweise oder komplett aus diesem beantwortet werden können. Die Geschwindigkeit, mit der eintreffende Anfragen beantwortet werden, wird so im Laufe der Zeit optimiert.

Installation von Unbound

Ein schlanker und einfach zu konfigurierender Nameserver ist *Unbound*. Seine Stärke ist das Auflösen von Namen und das Cachen der Ergebnisse. Das Ausliefern von Zonendaten hingegen ist nur sehr begrenzt möglich. Wenn Sie auch auf diesem Gebiet experimentieren möchten, sollten Sie zu *Bind9* als

Nameserver greifen. Im Workshop werden wir mit Unbound arbeiten, ein Austausch ist aber jederzeit möglich.

Installiert wird Unbound über die Paketverwaltung:

```
root@fuzzball:~# apt-get install unbound
Reading package lists... Done
✂ After this operation, 2,261 kB of additional disk space ↩
     will be used.
Do you want to continue [Y/n]? y
✂ Setting up unbound (1.4.16-1) ...
 * Starting recursive DNS server unbound [ OK ]
```

Konfiguration von Unbound

Der Nameserver soll sowohl für die Hosts am Link, als auch für *fuzzball* selbst, zuständig sein. Er muss daher auf der Loopback Address und auf einer der Adressen von Interface *eth1* lauschen. Auf den ersten Blick mag sich da die Link-local Address anbieten, schließlich befindet sich keiner der potentiellen Benutzer außerhalb des lokalen Scopes. Leider akzeptieren nicht alle Betriebssysteme einen Resolving Nameserver mit einer Link-local Address. Manche stören sich auch an der Schreibweise mit Interface hinter der Adresse: `fe80::1%eth0`. Es bleibt uns also nichts anderes übrig, als eine Global Unicast Address zu verwenden. Der Dienst soll auf dem Standard-Port 53 angeboten werden. Eine Zugriffskontrolle erlaubt dem Präfix des Links *internal* den Zugriff auf den DNS-Server. Versuchen Sie sich ruhig selbst an der Konfiguration! Mit dem Kommando `man unbound.conf` erhalten Sie alle benötigten Informationen zur Konfigurationsdatei.

Unter Beachtung der eben genannten Anforderungen ergibt sich eine Konfiguration wie in Abbildung 5.19 gezeigt.

Sobald die Konfigurationsdatei geschrieben wurde, starten wir den Nameserver neu:

```
root@fuzzball:~# service unbound restart
 * Restarting recursive DNS server unbound [ OK ]
```

Nameserver testen

Mit dem Kommando `host` können wir uns von der Funktion des Nameservers überzeugen. Das erste Argument ist der aufzulösende Hostname, das zweite und optionale Argument benennt

Abbildung 5.19 Konfiguration von Unbound

Datei: `/etc/unbound/unbound.conf`

```
server:
        verbosity: 1

        interface: ::1
        interface: 2a01:198:200:8a23::1

        port: 53

        access-control: ::1/128 allow
        access-control: 2a01:198:200:8a23::/64 allow
```

den zu befragenden Nameserver. Wir werden natürlich unseren eigenen Nameserver befragen, weshalb wir ihn auf der Loopback Address ansprechen:

```
user@fuzzball:~$ host www.ipv6-workshop.de ::1
Using domain server:
Name: ::1
Address: ::1#53
✂ www.ipv6-workshop.de has IPv6 address ↩
     2001:67c:26f4:800::6:80
```

Nameserver festlegen

Wenn der Server wie erwartet antwortet, definieren wir ihn, falls das System es nicht bereits in vorrauseilendem Gehorsam getan hat, auf *fuzzball* als Standard-Nameserver:

```
root@fuzzball:~# resolvconf -u
root@fuzzball:~# cat /etc/resolv.conf
nameserver ::1
```

In der Datei `/etc/resolv.conf` sind unter Ubuntu GNU/Linux die Adressen der Nameserver des Systems hinterlegt. Das eben ausgeführte Kommando hat die Datei überschrieben und dabei die Loopback Address als neue Nameserver-Adresse festgelegt. Mit dem Kommando `cat` haben wir uns den Inhalt der Datei anzeigen lassen.

Erreichbarkeit testen

Die Erreichbarkeit des Nameservers testen wir sicherheitshalber auch von einem der Hosts aus. Zum Beispiel durch eine einfache Namensauflösung auf *lynx*:

```
user@lynx:~$ host www.ipv6-workshop.de 2a01:198:200:8a23::1
Using domain server:
Name: 2a01:198:200:8a23::1
Address: 2a01:198:200:8a23::1#53
www.ipv6-workshop.de has IPv6 address ⤸
     2001:67c:26f4:800::6:80
```

Die korrekte Auflösung zeigt uns, dass der Nameserver funktioniert und von den Hosts erreicht und verwendet werden kann.

Die RDNSS Option

Nameserver-Adressen lassen sich auch als Option im Router Advertisement unterbringen und so bequem auf dem Link verteilen. Die Option heißt *RDNSS Option*, ist in RFC 6106 [JPBM10] spezifiziert, und reiht sich ein in die lange Liste der ICMPv6-Optionen. Insgesamt bietet sich uns damit der gleiche Komfort wie beim Betrieb eines sehr einfach gehaltenen DHCP-Servers, nur eben ohne einen dedizierten DHCP-Server für derartige Informationen bereitstellen zu müssen.

RDNSS Option im Router Advertisement

Mit der Verteilung der Router Advertisements hatten wir Radvd beauftragt. Wir ergänzen die Konfigurationsdatei gemäß Abbildung 5.20 um die RDNSS Option.

Damit die Änderungen wirksam werden, ist ein Neustart von Radvd erforderlich:

```
root@fuzzball:~# service radvd restart
Stopping radvd: radvd.
Starting radvd: radvd.
```

Router Advertisement

Selbstverständlich schauen wir uns das neue Router Advertisement in Wireshark an. Auf *fuzzball* fangen wir eines auf Interface *eth1* auf. Mit dem notwendigen Vorgehen sind wir inzwischen bestens vertraut. In Abbildung 5.21 ist das Router Advertisement zu sehen.

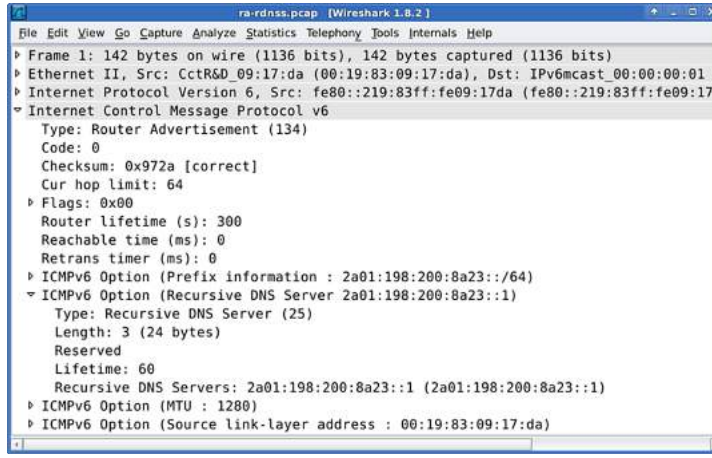
Neu hinzugekommen ist die ICMPv6-Option vom Typ 25. Sie enthält die Gültigkeitsdauer des Nameservers in Sekunden sowie die Adresse des Nameservers selbst. Die Gültigkeit mag auf den ersten Blick sehr kurz erscheinen, schließlich ändern sich die Adressen von Nameservern üblicherweise nicht jede

Abbildung 5.20 Konfiguration von Radvd mit RDNSS Address

Datei: `/etc/radvd.conf`

```
interface eth1 {
        AdvSendAdvert on;
        MinRtrAdvInterval 15;
        MaxRtrAdvInterval 60;

        AdvCurHopLimit 64;

        AdvManagedFlag off;
        AdvOtherConfigFlag off;
        AdvMobRtrSupportFlag off;
        AdvDefaultPreference medium;

        AdvDefaultLifetime 300;
        AdvReachableTime 0;
        AdvRetransTimer 0;

        AdvLinkMTU 1280;

        prefix 2a01:198:200:8a23::/64 {
                AdvOnLink on;
                AdvAutonomous on;
                AdvRouterAddr off;

                AdvValidLifetime 3600;
                AdvPreferredLifetime 1800;
        };

        RDNSS 2a01:198:200:8a23::1 { };

        AdvSourceLLAddress on;
};
```

Abbildung 5.21 Router Advertisement mit RDNSS Address



Minute. Wie so häufig, müssen wir auch hier wieder eine alte Denkweise abschütteln. Es ist unter IPv6 keine Seltenheit, wenn ein Link mit mehreren Routern ausgestattet ist. Und jeder Router preist unter Umständen seine ganz eigenen Nameserver an. Verlässt ein Router den Link, sollten auch alle Konfigurationsvariablen die er zuvor verteilt hat, zeitnah an Gültigkeit verlieren. Als Höchstwert für die RDNSS-Lifetime wird das doppelte maximale Sendeintervall (*MaxRtrAdvInterval*) empfohlen. Mit 60 Sekunden sind wir unter diesem empfohlenen Höchstwert geblieben.

Betriebssysteme und die RDNSS Option

Die hostseitige Unterstützung für die RDNSS Option ist leider nicht in allen Betriebssystemen vorhanden. Die Betriebssysteme iOS und OS X von Apple geben ein gutes Bild ab, was die Unterstützung der RDNSS Option angeht. Googles Android funktionierte zwar schon sehr früh gut mit IPv6, wertete aber auch Anfang 2013 noch nicht die RDNSS Option aus.

RDNSS Option unter Debian GNU/Linux

Unter Linux extrahiert das von vielen Distributionen und Desktop-Umgebungen verwendete Programm NetworkManager die Nameserver-Adressen aus den Router Advertisements. Anschließend fügt es die Informationen in die Datei `/etc/resolv.conf` des jeweiligen Systems ein. Wir hatten den NetworkManager auf *lynx* in Abschnitt 4.1 *Debian GNU/Linux 6* vorläufig entmachtet. Jetzt wo das Netz in seinen Grundzügen steht, darf der NetworkManager wieder die Kontrolle über die Interfaces auf *lynx* übernehmen.

Wir starten ihn wieder:

```
root@lynx:~# /etc/init.d/network-manager start
Starting network connection manager: NetworkManager.
```

Damit der NetworkManager auch nach einem Neustart noch arbeitet, fügen wir ihn wieder in die Boot-Sequenz ein.

```
root@lynx:~# update-rc.d network-manager enable
update-rc.d: using dependency based boot sequencing
```

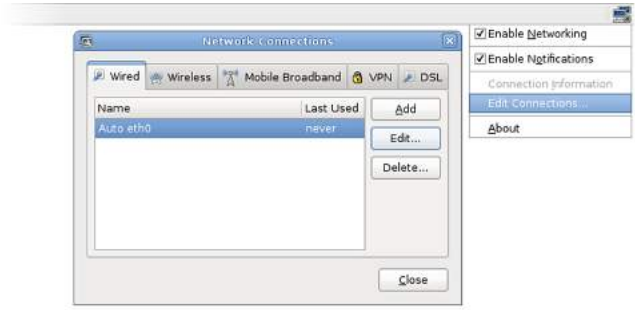


Abbildung 5.22 NetworkManager: Einstellungen und Profile

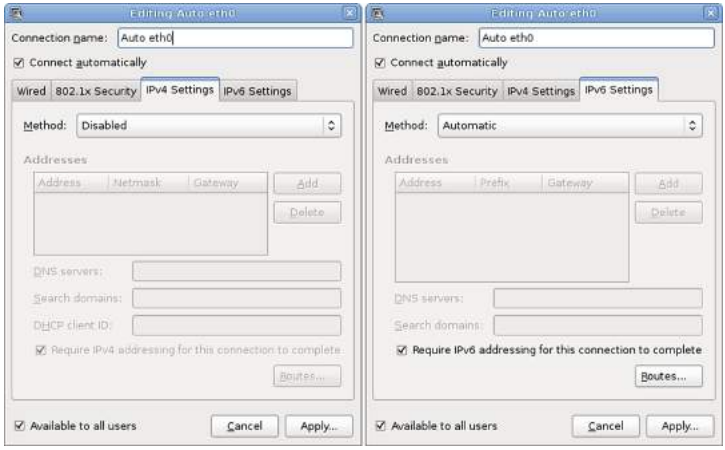


Abbildung 5.23 Profil *Auto eth0* auf *lynx*

Konfiguration durch NetworkManager

Auf dem Desktop befindet sich der NetworkManager in der oberen Leiste. Ein Symbol bestehend aus zwei verbundenen Computern bietet Zugriff auf die Oberfläche des Programms. Mit einem Linksklick werden die verfügbaren Profile angezeigt und mit einem Rechtsklick gelangt man in das Konfigurationsmenü. In letzteres begeben wir uns und wählen dann den Punkt *Edit Connections* (Abbildung 5.22) aus.

Nun öffnet sich eine nach Interfaces sortierte Liste verfügbarer Profile. Im Tab *Wired* suchen wir ein Profil zum Interface *eth0*, der Name könnte zum Beispiel *Auto eth0* lauten. Mit einem Klick auf *Edit* gelangen wir zu den Einstellungen des Profils (Abbildung 5.23).

Die Einstellungen für IPv4 stellen wir auf *Disabled*, die für IPv6 auf *Automatic*. Weitere Angaben sind nicht nötig für unseren Link, denn dort verteilen wir alle wichtigen Informatio-

nen mit den Router Advertisements. Wir schließen die offenen Dialoge und aktivieren das eben bearbeitete Profil mit einem Linksklick auf das NetworkManager-Symbol. Das Symbol zeigt für einen kurzen Augenblick Aktivität an und beruhigt sich wieder, sobald die Konfiguration erfolgreich verlaufen ist.

Überprüfen der Konfiguration

Davon werden wir uns überzeugen und lassen uns die Adressen des Interfaces anzeigen:

```
user@lynx:~$ ip addr show dev eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 ⤸
     qdisc pfifo_fast state UP qlen 1000
    link/ether 00:00:00:60:0d:1e brd ff:ff:ff:ff:ff:ff
    inet6 2a01:198:200:8a23:200:ff:fe60:d1e/64 scope ⤸
         global dynamic
       valid_lft 3600sec preferred_lft 1800sec
    inet6 fe80::200:ff:fe60:d1e/64 scope link
       valid_lft forever preferred_lft forever
```

Es zeigt sich das gewohnte Bild, die Autoconfiguration hat korrekt gearbeitet. Mit dem Kommando `cat` lassen wir uns den Inhalt der Datei `/etc/resolv.conf` anzeigen:

```
user@lynx:~$ cat /etc/resolv.conf
# Generated by NetworkManager
nameserver 2a01:198:200:8a23::1
```

Der NetworkManager hat die Nameserver-Adresse richtig aus dem Router Advertisement extrahiert und sie dem System bekannt gemacht. Der obligatorische Test beweist das Funktionieren der Namensauflösung und des Routings auf *lynx*:

```
user@lynx:~$ ping6 -c 3 ping.ipv6-workshop.de
PING ping.ipv6-workshop.de (ping.ipv6-workshop.de) 56 ⤸
     data bytes
64 bytes from ping.ipv6-workshop.de: icmp_seq=1 ttl=58 ⤸
     time=44.8 ms
✂ 3 packets transmitted, 3 received, 0% packet loss, time ⤸
     2002ms
```

RDNSS Option unter Windows 8

Windows 8 unterstützt die RDNSS Option leider nicht von Haus aus. Es gibt ein Programm namens *rdnssd-win32* welches die Funktionalität nachrüstet, dieses stammt aber nicht

Abbildung 5.24 Einstellungen von *LAN1* aufrufen

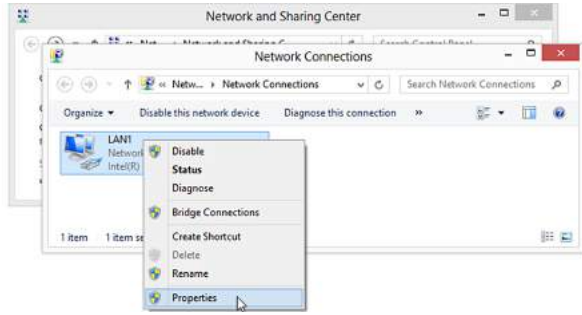
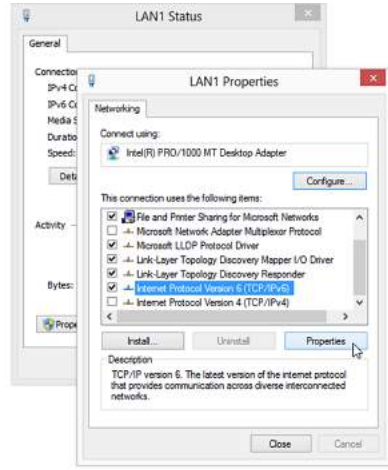


Abbildung 5.25 Protokolle von *LAN1*



von offizieller Seite.[5] Die Hoffnungen ruhen auf einer Nachrüstung durch eines der kommenden Service Packs. Es bleibt uns nichts anderes übrig, als die Adresse des Nameservers manuell einzutragen.

Dazu öffnen wir das *Connection and Sharing Center* und wählen dann *Change adapter settings* (Abbildung 5.24).

Ein Rechtsklick auf den Adapter *LAN1* und dann auf *Properties* bringt uns zur Liste der Protokolle. In der Protokollliste, dargestellt in Abbildung 5.25, ist IPv4 deaktiviert und IPv6 aktiviert. Wir wählen IPv6 aus und öffnen mit einem Klick auf *Properties* den Dialog zur IPv6-Konfiguration.

[5] http://sourceforge.net/projects/rdnssd-win32/

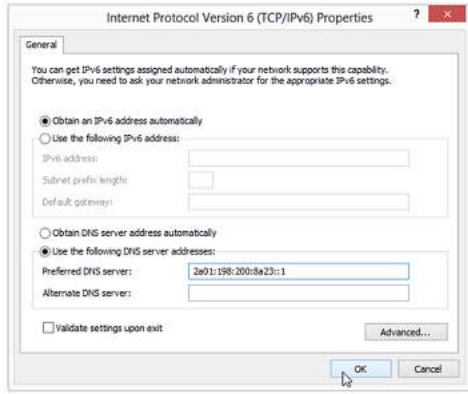


Abbildung 5.26 IPv6-Konfiguration von *LAN1*

Abbildung 5.27 Website des KAME-Projekts

Während die Einstellungen für die Adresse unverändert auf *automatically* stehen bleiben, erfordern die Nameserver-Einstellungen unser zutun. Wie in Abbildung 5.26 gezeigt, tragen wir die Nameserver-Adresse ein.

Bitte beachten Sie: Ihre Nameserver-Adresse wird anders lauten als jene in den Abbildungen. Die korrekte Adresse können Sie Ihrer Radvd-Konfiguration entnehmen.

Nach der Konfiguration schließen wir alle noch offenen Dialoge und schreiten zum Test der Einstellungen.

Überprüfen der Konfiguration

Zur Abwechslung besuchen wir diesmal die Website des KAME-Projekts unter *http://www.kame.net*. Mit der Namensauflösung und der tanzenden Schildkröte (Abbildung 5.27) ist der Nachweis erbracht, dass auch auf *felis* alle Einstellungen korrekt vorgenommen wurden.

6 | Der Übergang

So schön die Welt von IPv6 auch ist, wir werden zumindest vorübergehend noch Zugang zum IPv4-Internet brauchen. Der Übergang von IPv4 zu IPv6 wird deshalb von Übergangstechnologien begleitet. Diese helfen uns, in beiden Welten vertreten zu sein. Man teilt die Übergangstechnologien in drei Klassen ein:

- Parallelbetrieb
- Tunnelmechanismen
- Übersetzungsverfahren

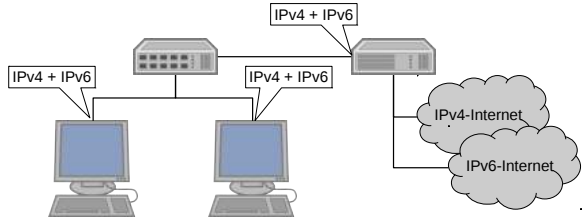
In jeder dieser Klassen tummeln sich zahlreiche Technologien. Oft unterscheiden sie sich nur im Detail. Wir werden nur die wichtigsten hier behandeln um anschließend eine geeignete für unseren internen Link auswählen zu können.

6.1 | Parallelbetrieb

Dual Stack

Unter Dual Stack versteht man den Parallelbetrieb von IPv4 und IPv6 am selben Link. Ein Interface kann ohne Probleme mehrere IPv4- und IPv6-Adressen aufnehmen. Dual Stack ist die einfachste Methode, IPv6 einzuführen, da es einfach zusätzlich zu IPv4 am Link ausgerollt wird. Ein kleines Szenario mit Dual Stack ist in Abbildung 6.1 zu sehen. Bis zu einer Übergabe an ein anderes Netz, zum Beispiel an einen Internet Service Provider, wird auf allen Links und Routern Dual Stack

Abbildung 6.1
Szenario
Dual Stack



gefahren. Ein Problem löst auch Dual Stack nicht: Die Knappheit der IPv4-Adressen. In Netzen, in denen IPv4-Adressen bereits jetzt rar sind, bietet sich Dual Stack daher nur eingeschränkt als Lösung an.

Bewertung von Dual Stack

Vorteile:

- Einfache Migration, viele Betriebssysteme können Dual Stack.
- Alle bestehenden Dienste können weiterhin unter gewohnter Adresse erreicht werden.

Nachteile:

- Das Problem der Adressknappheit bleibt bestehen.
- Doppelter Administrationsaufwand, da doppelte Filterregeln und Access Control Lists zu führen sind.

Dual Stack Lite

Unter Dual Stack Lite versteht man eine Kombination aus *Carrier Grade NAT* (CGN) und einem Tunnelmechanismus. Dual Stack Lite ist unter Zugangsanbietern ein beliebtes Mittel, um auch nach einer IPv6-Migration noch eine Grundversorgung mit IPv4 zu gewährleisten. Das Szenario von Dual Stack Lite ist in Abbildung 6.2 zu sehen. Das CPE des Kunden erhält natives IPv6 und eine IPv4-Adresse aus einem speziellen, in RFC 6598 [WKD+12] festgelegten, Bereich. Der Provider nutzt NAT/PAT um die vielen IPv4-Adressen der Kunden auf die wenigen verfügbaren IPv4-Adressen abzubilden. Da das CPE des Kunden üblicherweise auch ein NAT/PAT für IPv4 betreibt, werden IPv4-Datagramme am Ende mindestens zwei Mal übersetzt. Die Infrastruktur des Providers wird zu diesem Zeitpunkt bereits nativ mit IPv6 betrieben. Zusätzlich muss das CPE also die IPv4-Datagramme auch

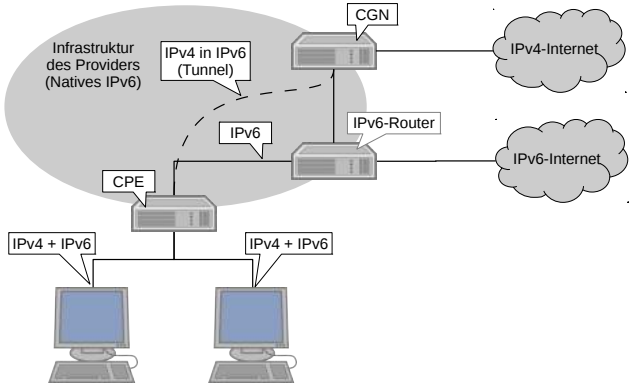


Abbildung 6.2 Szenario Dual Stack Lite

noch durch einen Tunnel durch die IPv6-Infrastruktur befördern.

Bewertung von Dual Stack Lite

Vorteile:

- Der Provider spart wertvolle Adressen ein.
- Die Infrastruktur basiert zukunftssicher auf IPv6.

Nachteile:

- Der Kunde erhält keine öffentliche IPv4-Adresse mehr.
- Ein Port-Forwarding muss vom Provider konfiguriert werden.

6.2 | Tunnelmechanismen

Tunnelmechanismen basieren darauf, Pakete des einen Internetprotokolls als Payload in Paketen des anderen Internetprotokolls zu transportieren. Die großen Unterschiede finden sich im Aufbau und Betrieb des Tunnels. Das Spektrum reicht von statischer Konfiguration der Tunnelendpunkte bis hin zu dynamischen, automatisch konfigurierten Tunneln.

Bewertung der Tunnelmechanismen

Folgende grundlegenden Vor- und Nachteile teilen sich aber alle Tunnel:

Vorteile:

- Keine Änderung an bestehender Infrastruktur notwendig.
- Schnelle Verfügbarkeit eines zweiten Internetprotokolls.

Nachteile:

- Komplexität des Netzes steigt und erschwert die Fehlersuche.
- Probleme des unterliegenden Protokolls beinträchtigen auch das getunnelte Protokoll.

4in6

Bei *4in6* handelt es sich, wie der Name schon vermuten lässt, um ein Tunneling von IPv4 in IPv6. IPv4-Datagramme werden als Payload in IPv6-Paketen platziert. Aus Sicht der IPv4-Endpunkte des Tunnels, bildet IPv6 den Link-layer. Die Konfiguration der Endpunkte geschieht statisch. Das Prinzip ist in RFC 2473 [CD98] beschrieben.

6in4

Das Tunneling von *6in4* ist dem vorangegangen Mechanismus sehr ähnlich. Diesmal dient jedoch die IPv4-Infrastruktur als Link-layer für IPv6. Eingerichtet wird der Tunnel ebenfalls durch statische Konfiguration. 6in4 basiert ebenfalls auf den in RFC 2473 [CD98] beschriebenen Verfahren.

6over4

Etwas komplizierter gestaltet sich *6over4*. Es ist ein Mechanismus zum Transport von IPv6-Datenpaketen zwischen Dual-Stack Knoten, welche über eine IPv4-Infrastruktur miteinander verbunden sind. Voraussetzung ist eine funktionierende IPv4-Multicast-Infrastruktur, da hier eine Art Neighbor Discovery über IPv4-Multicast durchgeführt wird. Damit sind die Nodes in der Lage, sich selbst zu finden. Eine manuelle Tunnelkonfiguration ist nicht erforderlich. Die Notwendigen Adressen für die IPv4-Interfaces werden gebildet, indem die IPv4-Adresse eines Nodes in hexadezimaler Schreibweise und das Präfix `fe80::/64` miteinander kombiniert werden. Das Vorgehen ist in Abbildung 6.3 dargestellt.

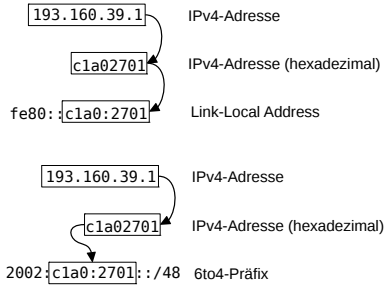


Abbildung 6.3 6over4 Link-local Address

Abbildung 6.4 6to4-Präfix berechnen

6to4

Bei *6to4* werden IPv6-Pakete in IPv4-Datagrammen über eine IPv4-Infrastruktur transportiert. Der Mechanismus ist in RFC 3056 [CM01] beschrieben. Jeder öffentlichen IPv4-Adresse steht ein Präfix der Länge /48 zur Verfügung. Die Bildung des Präfixes erfolgt gemäß Abbildung 6.4. Zwei Kommunikationspartner, welche beide 6to4 nutzen, können über eine vorhandene IPv4-Infrastruktur Daten austauschen. Die IPv4-Adresse des zuständigen Routers der Gegenseite, können sie einfach dem 6to4-Präfix entnehmen.

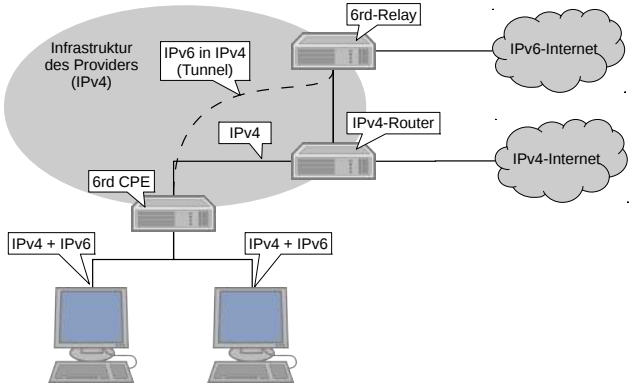
Komplizierter wird es, wenn einer der Kommunikationspartner nicht 6to4 nutzt. Dann springen die sogenannten *Relay-Server* ein. Sie announcen auf ihren IPv6-fähigen Interfaces das allgemeine 6to4-Präfix `2002::/16`, und erhalten so IPv6-Pakete, die an 6to4-Nodes adressiert sind. Die Relay-Server packen die empfangenen Pakete in IPv4 ein, und schicken diese an die IPv4-Adresse aus dem Präfix. *6to4-Nodes* wiederum erreichen die Relay-Server über die Anycast Address `192.88.99.1`.

Die automatische Konfiguration und das Vorhandensein öffentlicher 6to4 Relay Server machen 6to4 zu einer bequemen Tunneling-Methode. Einen Nachteil hat der Mechanismus dennoch: Die Betreiber der 6to4 Relay Server könnten jeden über ihre Server abgewickelten Datenverkehr mitlesen.

6rd

6rd, auch *IPv6 rapid deployment* genannt, ist eine spezielle Form von 6to4. Entwickelt wurde der Mechanismus beim französischen Provider *Free*, dessen Kunden schon früh auf eine Versorgung mit IPv6 drängten. Innerhalb von nur fünf Wo-

Abbildung 6.5
Szenario 6rd



chen wurde der Mechanismus ausgerollt. Im Unterschied zu 6to4 verwendet 6rd ein allgemeines Präfix des Providers. An dieses werden die IPv4-Adressen des Kunden, ebenfalls aus dem Adressbereich des Providers, angefügt. Die *Relay-Server* werden vom Provider selbst betrieben, deshalb gelten sie als vertrauenswürdig. Eine Aktualisierung der Software auf dem CPE der Kunden ist bei der Einführung von 6rd unumgänglich. Der Provider, dessen Infrastruktur weiterhin auf IPv4 basierte, konnte so Zeit für eine wohlüberlegte Migration gewinnen. Die Technologie wurde von Free in RFC 5969 [TT10] beschrieben.

AYIYA Gleich mehrere Probleme auf einmal löst das Protokoll AYIYA, auch unter *Anything In Anything* bekannt. Der Transport geschieht bewusst nicht direkt in IPv4, sondern in einem Upper Layer Protocol. Letztere kommen in der Regel problemlos durch NAT/PAT-Router oder CGN. AYIYA erlaubt es, mehrere Tunnel gleichzeitig hinter einem NAT/PAT zu betreiben. Darüber hinaus kann ein Tunnelendpunkt auch seine Adresse ändern, was dazu führt, dass selbst periodisch wechselnde IPv4-Adressen die Stabilität des Tunnels nicht beeinträchtigen können.

Typischerweise werden IPv6-Pakete in UDP verpackt, die dann wiederum mit IPv4 auf die Reise geschickt werden. Die Gegenseite ist dann üblicherweise der Server eines Tunnelbrokers.

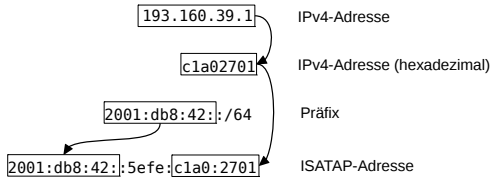


Abbildung 6.6 ISATAP-Adresse berechnen

SixXS, in dessen Reihen AYIYA entwickelt wurde, ist der bekannteste Nutzer dieses Protokolls. Die Spezifikation existiert bisher nur als Entwurf, kann aber auf der Website des SixXS-Projektes heruntergeladen werden.

ISATAP

Das *Intra-Site Automatic Tunnel Addressing Protocol* ist eine Gemeinschaftsentwicklung von Microsoft, Cisco und Boeing. Es kombiniert die von 6to4 und 6over4 bekannten Mechanismen, um eine Global Unicast Address für einen Node zu erzeugen, welcher nur IPv4-Konnektivität besitzt. Dafür ist die Zuteilung eines Präfixes durch den zuständigen Provider notwendig. Die Adresse berechnet sich dann wie in Abbildung 6.6 gezeigt. Es existieren Implementierungen für alle gängigen Betriebssysteme von Microsoft, für Linux und einige Varianten von Cisco IOS. Der genaue Ablauf der automatischen Konfiguration ist in RFC 5214 [TGT08] spezifiziert.

Teredo

Teredo ist eine Entwicklung aus dem Hause Microsoft und ist so genial wie kompliziert. Es ist in der Lage, die verschiedensten Arten von NAT/PAT zu überwinden. Die Teredo-Software, in aktuellen Versionen von Windows bereits eingebaut, stellt ein Interface mit einer IPv6-Adresse zur Verfügung. Die Adresse eines Teredo-Interfaces wird nach dem in Abbildung 6.7 gezeigten Schema gebildet. Der Standard sieht vor, dass Teile der Adresse verschleiert werden, indem die Bits der entsprechenden Teile invertiert werden. Öffentliche *Teredo-Server* und *Teredo-Relays* im Internet erledigen die Konfiguration des Tunnels und wickeln den Datenverkehr ab. Gleichzeitig sorgen sie auch dafür, dass die Verbindung in zwischengelagerten NAT/PAT-Routern nicht abläuft, indem sie regelmäßig sogenannte *Bubbles* versenden. Die Komplexität von Teredo lässt sich bei Betrachtung des Szenarios in Abbildung 6.8 erah-

Abbildung 6.7
Teredo-Adresse berechnen

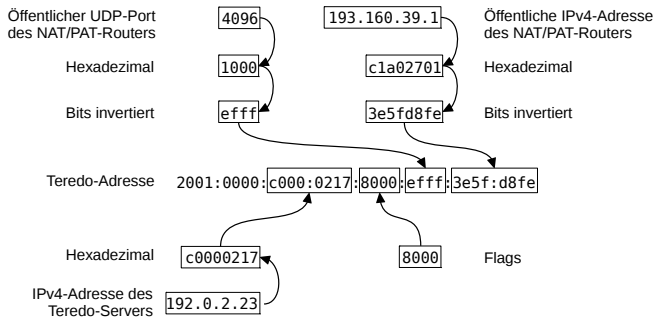
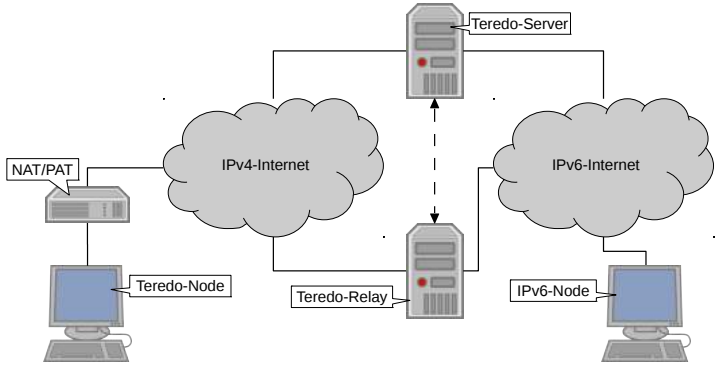


Abbildung 6.8
Szenario Teredo



nen. Wenn ein Teredo-Relay Daten für einen *Teredo-Node* hat, informiert es den zuständigen Teredo-Server darüber, dessen IPv4-Adresse es der Teredo-Adresse des Nodes entnehmen kann. Der Teredo-Server informiert dann den Teredo Node über die bestehende, mit Bubbles aktiv gehaltene, Verbindung über anstehende Daten. Der Teredo-Node baut daraufhin von innen heraus eine Verbindung zum Teredo-Relay auf, welches auf dieser Verbindung dann die Daten ausliefert. Ein einfaches IPv6-Paket, adressiert an einen Teredo-Node, führt dazu, dass dieser die NAT/PAT-Router auf dem Weg zum Teredo-Relay von innen heraus *aufbohrt*.[1] Die vermeintliche Sicherheit, die eine NAT/PAT-Installation unter IPv4 noch zu bieten schien, wird nicht zuletzt von Teredo ausgehebelt. Wenn sich Nodes mit Teredo in einem IPv4-Netz befinden, muss davon ausgegangen werden, dass IPv6-Pakete

[1] Der Schiffsbohrwurm Teredo portoricensis stand wohl nicht umsonst bei der Namensgebung Pate.

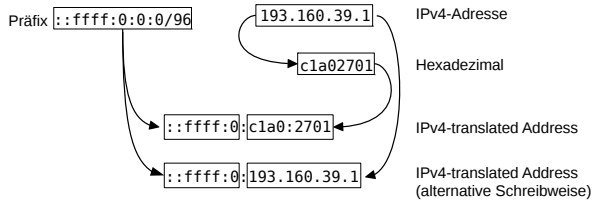


Abbildung 6.9 SIIT-Adresse berechnen

bis zum Node vordringen können. Auf natives IPv6 zu verzichten, kann also schlimmstenfalls zu weniger Sicherheit führen.

Das Studium des Teredo-Standards, hinterlegt in RFC 4380 [Hui06], sei dem geneigten Leser empfohlen.

6.3 | Übersetzungstechnologien

Die Übersetzungstechnologien verzichten auf das Verpacken von Paketen in anderen Paketen, sondern gehen einen Schritt weiter. Dabei wird der Header des einen Internetprotokolls analysiert, um aus den gewonnenen Informationen einen möglichst ähnlichen Header des anderen Internetprotokolls zu erstellen. Felder wie die Protokollnummer oder das Hop Limit lassen sich dabei recht problemlos übersetzen. Schwieriger wird es bei Adressen, fragmentierten Daten und ICMP beziehungsweise ICMPv6 Messages.

Bewertung von Übersetzungstechnologien

Vorteile:

- Es wird nur ein Internetprotokoll im Netz gesprochen.
- Weniger Komplexität in der Infrastruktur.

Nachteile:

- Nicht alle Upper Layer Protocols lassen sich übersetzen.

SIIT

Die *Stateless IP/ICMP Translation* basiert auf den IPv4-translated Addresses. Diese haben das Präfix `::ffff:0:0:0/96` und werden, wie in Abbildung 6.9 gezeigt, erzeugt. Genutzt werden kann SIIT für die Anbindung von IPv6-Nodes an das IPv4-Internet. Das Präfix wurde so gewählt, dass die übersetzte Adresse die Prüfsumme nicht verändert. Die Prüfsumme

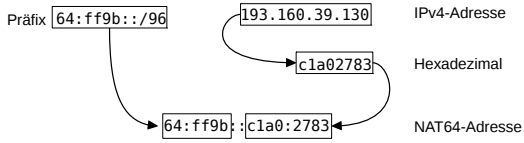


Abbildung 6.10 NAT64-Adresse berechnen

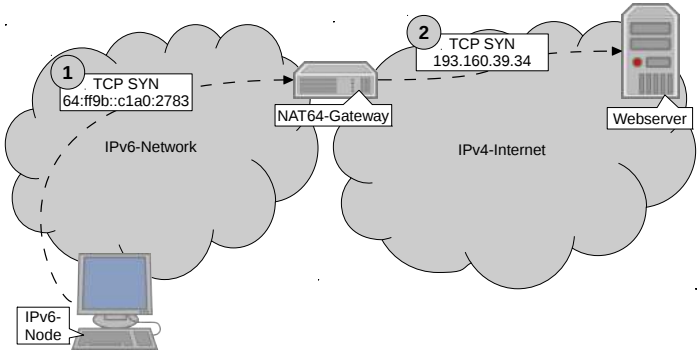


Abbildung 6.11 Szenario NAT64

des Pseudo-Headers eines Upper Layer Protocols muss daher nicht neu berechnet werden. Der Standard zu SIIT, spezifiziert in RFC 6145 [LBB11], behandelt weder die Themen Adressvergabe noch das Routing.

NAT64

Mit *NAT64* existiert eine Technologie, mit der auch in Netzen mit nativem IPv6 auf das IPv4-Internet zugegriffen werden kann. Der IPv4-Adressraum lässt sich ohne Verluste im IPv6-Adressraum abbilden. Wie schon bei SIIT, wird auch bei NAT64 von dieser Möglichkeit Gebrauch gemacht. NAT64 hat dafür ein eigenes Well-known-Präfix, nämlich `64:ff9b::/96`, in RFC 6052 [BHB+10] und RFC 6146 [BMvB11] zugeteilt bekommen. Alternativ zum Well-known-Präfix, kann auch ein Präfix aus dem eigenen Adressbereich genutzt werden. Die Berechnung der Adressen erfolgt wie in Abbildung 6.10 gezeigt. Für den Betrieb von NAT64 benötigt man ein *NAT64-Gateway*, manchmal auch *NAT64-Server* genannt, das als Brücke zwischen der IPv6-Welt und der IPv4-Welt agiert. Das NAT64-Gateway braucht dafür mindestens eine IPv4-Adresse. Idealerweise erhält das NAT64-Gateway ein ganzes Subnetz, damit es keine Verbindungen wegen Adressknappheit ablehnen muss. Betrachten wir das NAT64-Szenario in Abbildung 6.11 etwas genauer. Der IPv6-Node möchte mit

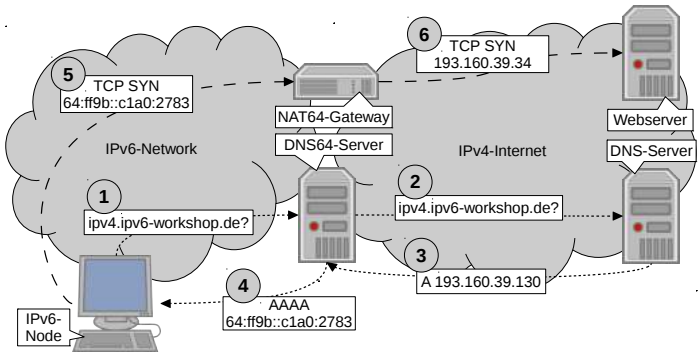


Abbildung 6.12 Szenario NAT64 und DNS64

einem Webserver kommunizieren, der nur über IPv4 erreichbar ist. Der IPv6-Node hingegen befindet sich in einem Netz, in dem es keine IPv4-Adressen mehr gibt. Darum nutzt er NAT64, indem er die IPv4-Adresse in eine NAT64-Adresse einbettet und mit dieser kommuniziert. Das NAT64-Gateway übersetzt das Paket, hier war es ein TCP SYN im Upper Layer Protocol, und schickt es an die IPv4-Adresse. Die IPv4-Adresse konnte das NAT64-Gateway der NAT64-Adresse entnehmen.

Diese Übersetzungstechnologie erlaubt es uns, Netze ohne IPv4 zu planen, und dennoch mit dem langsam veraltenden IPv4-Internet in Kontakt zu bleiben. Einen kleinen Haken hat diese Technologie aber: Wenn auf Protokollebene über IP-Adressen gesprochen wird, zum Beispiel beim *File Transfer Protocol* (FTP), reden die Kommunikationspartner aneinander vorbei. Mit den NAT64-Adressen wird ein IPv4-Node nicht viel anfangen können.

NAT64 und DNS64

So richtig punkten kann NAT64 aber erst im Zusammenspiel mit *DNS64*. Mit DNS64 meint man einen DNS-Proxy oder speziellen DNS-Server, der IPv4-Adressen in NAT64-Adressen umwandelt. Antworten auf Anfragen, die lediglich A-Einträge aufweisen, werden vom DNS64-Server manipuliert. Dazu werfen wir einen Blick auf das um DNS64 erweiterte NAT64-Szenario in Abbildung 6.12. Auf dem IPv6-Node möchte ein Nutzer eine Website ansurfen, die nur über IPv4 erreichbar ist. Er schickt den Hostnamen zwecks Auflösung an seinen Re-

solving DNS Server, im Szenario ist dies natürlich ein DNS64-Server. Der DNS64-Server löst den Namen auf, und stellt dabei fest, dass kein AAAA-Eintrag zurückkommt. Daher schreibt er die IPv4-Adresse aus dem A-Eintrag in eine NAT64-Adresse um. Die umgeschriebene Adresse wird dem IPv6-Node als AAAA-Eintrag zum angefragten Hostnamen präsentiert.[2] Die restlichen Abläufe entsprechen denen aus dem vorangegangenen Szenario.

6.4 | NAT64 und DNS64 am Link

Die Entscheidung

Von den hier vorgestellten Übergangstechnologien nutzen wir eine bereits aktiv. Unser Tunnel zu SixXS basiert auf AYIYA. Die Erfahrung zeigt, dass während einer Migration stets zwei oder mehr Übergangstechnologien miteinander kombiniert werden. In unserem Netz würden sich NAT64 und DNS64 eigentlich ganz gut machen. Dann müssten wir kein Protokoll ausrollen dessen Tage schon bald als gezählt gelten könnten. Trotzdem verlören wir nicht den Kontakt zum IPv4-Internet, welches zumindest vorübergehend noch von großer Bedeutung ist.

Anforderungen

Unser Präfix hat eine Länge von 64 Bits, und bietet damit keinen Spielraum für ein separates NAT64-Präfix. Stände uns ein kürzeres Präfix (also mehr Adressen) zur Verfügung, zum Beispiel /48, könnten wir ein /64 IPv6-Netz für den Link reservieren, und ein /96 IPv6-Netz als NAT64-Präfix verwenden.[3] Daher müssen wir das Well-known-Präfix `64:ff9b::/96` für NAT64 verwenden.

Auf dem NAT64-Gateway benötigen wir eine IPv4-Adresse für ausgehende Verbindungen. Besser wäre ein größerer Adress-

[2] Es bleibt abzuwarten, wie sich dieses Verfahren bei flächendeckender Ausbreitung von *Domain Name System Security Extensions* (DNSSEC) auf die Gültigkeit von DNS-Antworten auswirkt.

[3] Alternativ können Sie auch ein /48 Präfix bei SixXS beantragen. Daraus können Sie dann, wie in den vorherigen Kapiteln gelernt, ein /64 Präfix am Link verteilen. Aus dem restlichen Bereich suchen Sie sich dann ein eigenes /96 Präfix für ihr NAT64-Gateway aus.

bereich, auch Adresspool genannt, damit dem NAT64-Gateway mehr Möglichkeiten offenstehen, wenn IPv6- zu IPv4-Adressen übersetzt werden müssen. Dazu kann man dem NAT64-Gateway einen Adresspool aus RFC 1918 [RMK+96] zuweisen. Die Adressen aus dem Adresspool werden dann vor dem Verlassen des Systems auf die eigentliche IPv4-Adresse des Systems per klassischem NAT/PAT umgeschrieben. Für jede IPv6-Adresse die mit dem IPv4-Internet kommunizieren möchte, wird eine IPv4-Adresse aus dem Adresspool benötigt. Ein /24 IPv4-Netz als Adresspool für übersetzte Verbindungen sollte genügen. Eine genaue Erklärung des Verfahrens zur Adresswahl beim Übersetzen findet sich in RFC 6052 [BHB+10].

Als NAT64-Gateway wird *fuzzball* dienen. Nicht nur, weil *fuzzball* bereits als Router dient, sondern auch, weil kein anderes System am Link eine funktionierende IPv4-Konnektivität besitzt. Ohne IPv4-Konnektivität aber kann ein NAT64-Gateway nicht funktionieren.

Tayga

Eine Software für NAT64-Gateways, die unsere Anforderungen erfüllt, ist *Tayga*.[4] Tayga läuft sehr schnell, braucht nach dem Starten keine root-Rechte mehr und lässt sich sehr einfachen konfigurieren. Da sich Tayga nicht im Kernel einnistet, sondern als normaler, unprivilegierter Prozess arbeitet, benötigt es eine Schnittstelle zum Austausch von IP-Paketen mit dem Kernel. Darum richtet Tayga beim Starten ein Interface mit dem Namen *nat64* ein. In dieses Interface gehen die Pakte vor der Übersetzung ein, werden übersetzt, und verlassen es als übersetzte Pakete. Eingehende IPv6-Pakte werden so zu IPv4-Datagrammen und umgekehrt.

NAT64 im Lern- und Bastelnetzwerk

Wo genau sich das Interface *nat64* inmitten der zahlreichen anderen Interfaces auf *fuzzball* einordnet, ist in Abbildung 6.13 zu sehen.

Unser Lern- und Bastelnetzwerk hat inzwischen beachtlich an Komplexität gewonnen. Betrachten wir zunächst die Maschi-

[4] *http://www.litech.org/tayga/*

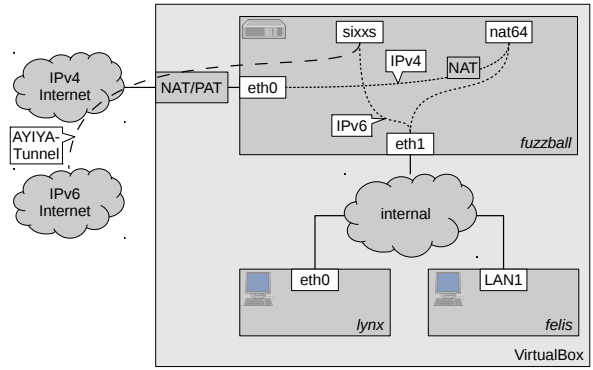


Abbildung 6.13 Maschinen und Interfaces in VirtualBox

nen *lynx* und *felis*. Beide haben jeweils ein Interface am Link *internal*. Präfixinformationen und Resolving DNS Server beziehen sie über diese Interfaces von *fuzzball*. Mit dem Interface *eth1* ist der Router am Link *internal* vertreten. IPv6-Pakete aus und in das IPv6-Internet leitet er über das Tunnelinterface *sixxs* weiter. Die Software hinter *sixxs* wiederum packt die IPv6-Pakete in IPv4-Datagramme ein, und schickt sie dann über das Interface *eth0* zum Tunnelserver des SixXS-Projektes. Ankommende Pakete durchlaufen den Prozess in umgekehrter Reihenfolge. Einen anderen Weg gehen IPv6-Pakete die an eine Adresse aus dem NAT64-Präfix adressiert sind. Über *eth1* gelangen sie in *fuzzball* hinein und werden dann an das Interface *nat64* durchgereicht (IPv6-Forwarding). Hinter *nat64* steht Tayga als verantwortliche Software. Tayga übersetzt die IPv6-Pakete in IPv4-Datagramme. Mit einer Quelladresse aus dem Adresspool verlassen diese das Interface *nat64*. Bevor sie über *eth0* ins IPv4-Internet geschickt werden können (IPv4-Forwarding), müssen sie noch einmal durch eine Network Address Translation. Der letzte Schritt ist notwendig, damit die Pooladressen nicht mehr in den Datagrammen stehen, wenn sie *eth0* verlassen. VirtualBox beispielsweise hat keine Route zum Adresspool, und wüsste deshalb nicht wohin mit potentiellen Antwortdatagrammen.

IPv4-Forwarding aktivieren

Zusätzlich zum bereits aktivierten IPv6-Forwarding benötigen wir auf *fuzzball* in Zukunft also auch IPv4-Forwarding. Dazu ergänzen wir die Kernelparameter in der Datei `/etc/sysctl.conf` entsprechend Abbildung 6.14.

Datei: `/etc/sysctl.conf` (Auszug)

```
1 # Uncomment the next line to enable packet forwarding for IPv6
2 # Enabling this option disables Stateless Address
3 # Autoconfiguration based on Router Advertisements for this host
4 net.ipv6.conf.all.forwarding=1
5
6 # Uncomment the next line to enable packet forwarding for IPv4
7 net.ipv4.ip_forward=1
```

Abbildung 6.14 Forwarding permanent aktivieren

Im Anschluss wenden wir die neuen Parameter auf den laufenden Kernel an:

```
root@fuzzball:~# sysctl -p
net.ipv4.ip_forward = 1
net.ipv6.conf.all.forwarding = 1
```

Damit sind die Grundlagen für die Nutzung von Tayga gelegt.

Installation von Tayga

Schreiten wir zur Installation von Tayga. Wie gewohnt beziehen wir die Software aus dem Repository des Betriebssystems:

```
root@fuzzball:~# apt-get install tayga
✂ The following NEW packages will be installed:
  tayga
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
✂ Setting up tayga (0.9.2-3) ...
 * tayga disabled, please adjust the configuration to your needs
 * and then set RUN to 'yes' in /etc/default/tayga to enable it.
```

Die Installation endet mit dem Hinweis auf eine unvollständige Konfiguration.

Konfiguration von Tayga

Die Konfiguration werden wir nun nachholen. Die Standardeinstellungen von Tayga sind für unsere Situation größtenteils zutreffend. Was wir noch festlegen müssen, ist das Präfix welches wir für die IPv4-Mapped Addresses verwenden werden. Im Workshop entscheiden wir uns für das Well-known-Präfix. Falls Sie bereits mehrere Präfixe auf *fuzzball* betrei-

Abbildung 6.15 Tayga Konfigurationsdatei

Datei: `/etc/tayga.conf` (Auszug)

```
# Configuration file for TAYGA 0.9.2

# TUN device that TAYGA will use to exchange IPv4 and IPv6
# packets with the kernel.  You may use any name you like,
# but 'nat64' is recommended.
tun-device nat64

# TAYGA's IPv4 address.  This is NOT your router's IPv4 address!
ipv4-addr 192.168.255.1

# TAYGA's IPv6 address.  This is NOT your router's IPv6 address!
ipv6-addr 2a01:198:200:8a23::64

# The NAT64 prefix.  The IPv4 address space is mapped into the
# IPv6 address space by prepending this prefix to the IPv4
# address.  Using a /96 prefix is recommended in most
# situations, but all lengths specified in RFC 6052 are
# supported.
prefix 64:ff9b::/96

# Dynamic pool prefix.
dynamic-pool 192.168.255.0/24

# Persistent data storage directory.
data-dir /var/spool/tayga
```

ben, können Sie auch ein Global-Unicast-Präfix aus Ihrem Bereich verwenden. Die Konfigurationsdatei sieht bei Nutzung des Well-known-Präfixes wie in Abbildung 6.15 gezeigt aus.

Die IPv4-Adressen können wir direkt übernehmen, auch der Adresspool ist hinreichend groß dimensioniert. Eine IPv6-Adresse aus dem Präfix, welches am Link *internal* anliegt, müssen wir allerdings Tayga opfern. Diese Adresse wird später vom Interface *nat64* genutzt werden. Der Interface Identifier der Adresse kann frei gewählt werden, zum Beispiel `::64`.

Sobald wir die Konfiguration von Tayga abgeschlossen haben, müssen wir dem Betriebssystem noch mitteilen, dass Tayga in Zukunft gestartet werden darf. Dafür setzen wir die Variable *RUN* im Init-Skript `/etc/default/tayga` auf *yes*. Die ande-

Datei: `/etc/default/tayga` (Auszug)

```
1  # Change this to "yes" to enable tayga
2  RUN="yes"
3
4  # Configure interface and set the routes up
5  CONFIGURE_IFACE="yes"
6
7  # Configure NAT44 for the private IPv4 range
8  CONFIGURE_NAT44="yes"
9
10 # Additional options that are passed to the Daemon.
11 DAEMON_OPTS=""
```

Abbildung 6.16 Tayga Initscript

ren Variablen im Init-Skript weisen das Betriebssystem an, uns einige Arbeitsschritte abzunehmen. Darunter die Network Address Translation für die Adressen aus dem Adresspool und die Interface-Konfiguration von *nat64*. Das gesamte Initscript zeigt die Abbildung 6.16.

Tayga starten

Nun steht dem Betrieb von Tayga nichts mehr im Wege. Wir starten Tayga:

```
root@fuzzball:~# service tayga start
 * Starting userspace NAT64 tayga
 Created persistent tun device nat64
```

Ein kurzer Blick auf die vorhandenen Interfaces zeigt uns, ob das Interface *nat64* korrekt erstellt wurde:

```
user@fuzzball:~$ ip link show
✂ 2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 ⤸
     qdisc pfifo_fast state UP qlen 1000
    link/ether 08:00:27:a5:6e:82 brd ff:ff:ff:ff:ff:ff
  3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 ⤸
     qdisc pfifo_fast state UP qlen 1000
    link/ether 00:19:83:09:17:da brd ff:ff:ff:ff:ff:ff

✂ 5: sixxs: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu ⤸
     1280 qdisc pfifo_fast state UNKNOWN qlen 500
    link/none
  6: nat64: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu ⤸
     1500 qdisc pfifo_fast state UP qlen 500
    link/none
```

NAT64 testen

Zum Testen von NAT64 werden wir einen Echo Request an einen der öffentlichen Nameserver von Google verschicken. Bei der Schreibweise der Adresse haben wir die Wahl:

Entweder wir ersetzen die letzten zwei Blöcke der NAT64-Adresse durch die IPv4-Adresse in Punknotation oder wir schreiben die IPv4-Adresse hexadezimal an dieselbe Stelle. Die IPv4-Adresse `8.8.8.8` wird als hexadezimal geschriebene NAT64-Adresse zu `64:ff9b::808:808`. Die alternative Schreibweise mit Punktnotation derselben Adresse lautet `64:ff9b::8.8.8.8`.

Der erste Test erfolgt von *fuzzball* aus:

```
user@fuzzball:~$ ping6 -c 3 64:ff9b::8.8.8.8
PING 64:ff9b::8.8.8.8 (64:ff9b::808:808) 56 data bytes
64 bytes from 64:ff9b::808:808: icmp_seq=1 ttl=46 ↩
    time=287 ms
✂ 3 packets transmitted, 3 received, 0% packet loss, time ↩
    1999ms
```

Die eintreffenden Antworten deuten darauf hin, dass Tayga korrekt arbeitet. Zur Sicherheit werden wir den gleichen Test auch von den Hosts aus durchführen. Als erstes von *lynx* aus:

```
user@lynx:~$ ping6 -c 3 64:ff9b::8.8.8.8
PING 64:ff9b::8.8.8.8 (64:ff9b::808:808) 56 data bytes
64 bytes from 64:ff9b::808:808: icmp_seq=1 ttl=46 ↩
    time=327 ms
✂ 3 packets transmitted, 3 received, 0% packet loss, time ↩
    2000ms
```

Und danach von *felis* aus:

```
C:\Users\user>ping -n 3 64:ff9b::8.8.8.8
Pinging 64:ff9b::808:808 with 32 bytes of data:
Reply from 64:ff9b::808:808: time=333ms
✂ Ping statistics for 64:ff9b::808:808:
    Packets: Sent = 3, Received = 3, Lost = 0 (0% loss),
```

Soeben haben zwei Hosts, die selbst keine IPv4-Konnektivität besitzen, mit einem Node im IPv4-Internet kommuniziert. Dies verdeutlicht, welch große Rolle NAT64 während und nach einer IPv6-Migration spielen könnte. Im Lern- und Bastelnetzwerk sorgt NAT64 bereits jetzt dafür, dass wir den Anschluss an das IPv4-Internet nicht aufgeben müssen. Denn IPv4 wird uns vermutlich noch eine ganze Weile begleiten.

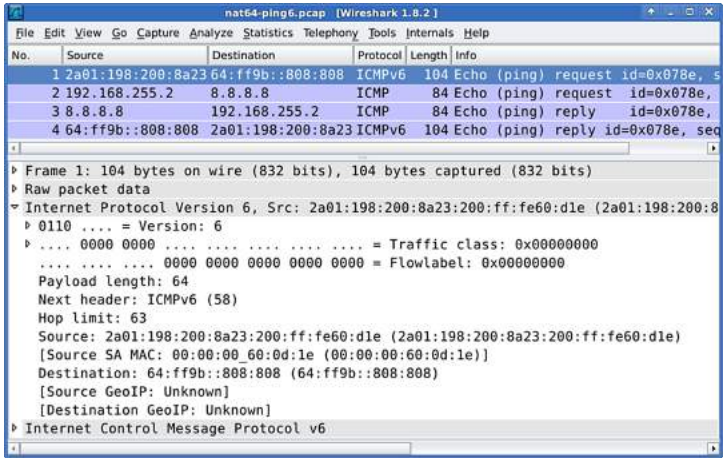


Abbildung 6.17 NAT64-Analyse: Paket 1

Übersetzung analysieren

Wie Tayga Pakete von einem in das andere Protokoll übersetzt, werden wir uns jetzt genauer anschauen. Wir werden uns allerdings darauf beschränken, die Übersetzung der IP-Header zu untersuchen, da sie die Grundlage der Übersetzung bilden. Dazu starten wir Wireshark auf *fuzzball* und belauschen das Interface *nat64*. Von *lynx* aus schicken wir wieder Echo Requests an die Adresse `64:ff9b::8.8.8.8`:

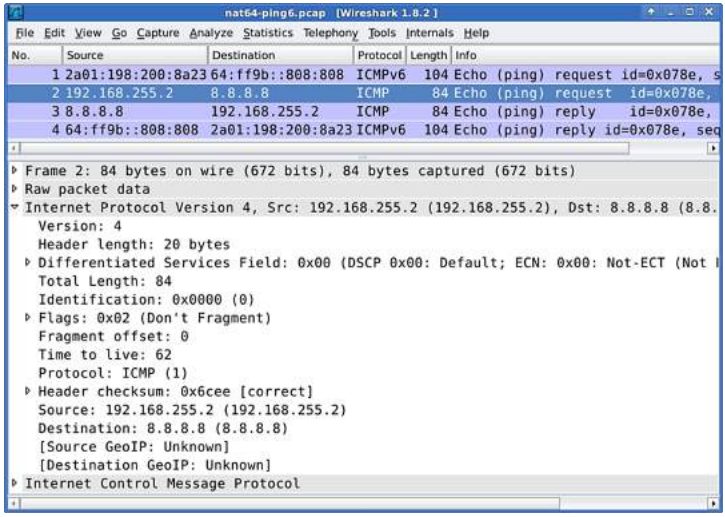
```
user@lynx:~$ ping6 -c 1 64:ff9b::8.8.8.8
PING 64:ff9b::8.8.8.8 (64:ff9b::808:808) 56 data bytes
64 bytes from 64:ff9b::808:808: icmp_seq=1 ttl=46 ⤸
     time=328 ms
✂ 1 packets transmitted, 1 received, 0% packet loss, time ⤸
     328ms
```

Nun stoppen wir den Mitschnitt und schauen uns die aufgefangenen Pakete an. Die Anzahl der Pakete hat sich erwartungsgemäß verdoppelt. Zu jedem der Echo Request und Echo Reply Messages in IPv6 existiert ein IPv4-Äquivalent.

Übersetzung von IPv6 nach IPv4

Betrachten wir zunächst die beiden IP-Header der Echo Request Messages aus den Abbildungen 6.17 und 6.18. Die Versions-Felder unterscheiden sich selbstverständlich. Im IPv4-Header wurde das Feld Header Length von Tayga mit der minimalen Headerlänge von 20 Bytes initialisiert. Das Feld Differentiated Services (IPv4) basiert auf dem Feld Traffic Class (IPv6) und enthält standardmäßig nur Nullen. Das Flow

Abbildung 6.18 NAT64-Analyse: Paket 2



Label (IPv6) lässt sich nicht in IPv4 abbilden, und fällt daher raus. Die Payload Length aus dem IPv6-Header musste für den IPv4-Header um die IPv4 Header Length erhöht werden. Der Wert hat im Feld Total Length (IPv4) Eingang gefunden. Die Fragmentierungsinformtionen im IPv4-Header musste Tayga selbst erstellen, da im IPv6-Paket kein Fragment Extension Header als Informationsquelle zur Verfügung stand. Tayga betrachtet die Übersetzung als einen Hop und hat daher das Hop Limit dekrementiert. Im IPv4-Header heißt das dazugehörige Feld Time To Live und wird mit dem dekrementierten Wert versehen. Bei IPv6 gibt der Next Header Auskunft über das nachfolgende Protokoll. Auf Basis dieser Information hat Tayga die Protokollnummer ermittelt und sie im IPv4-Header im Feld Protocol vermerkt. Die bei IPv4 noch gebräuchliche Prüfsumme (Feld Header Checksum) musste neu erstellt werden. Als Quelladresse setzt Tayga eine der Adressen aus dem Adresspool in den IPv4-Header ein. Die Zieladresse wurde aus der NAT64-Adresse des Ziels extrahiert.

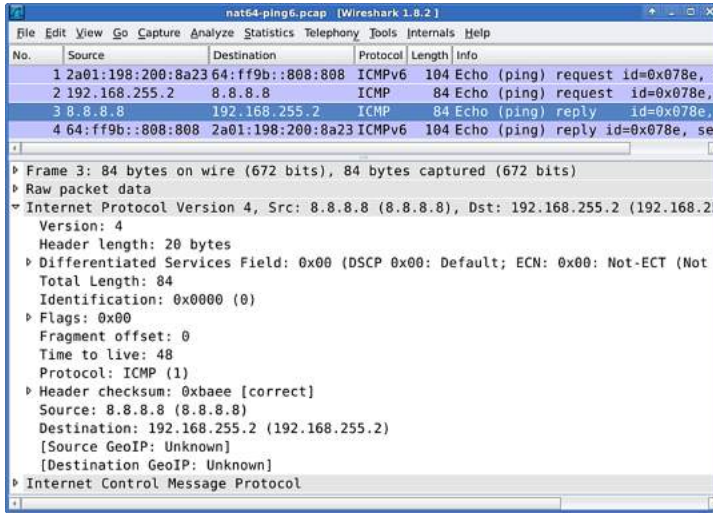
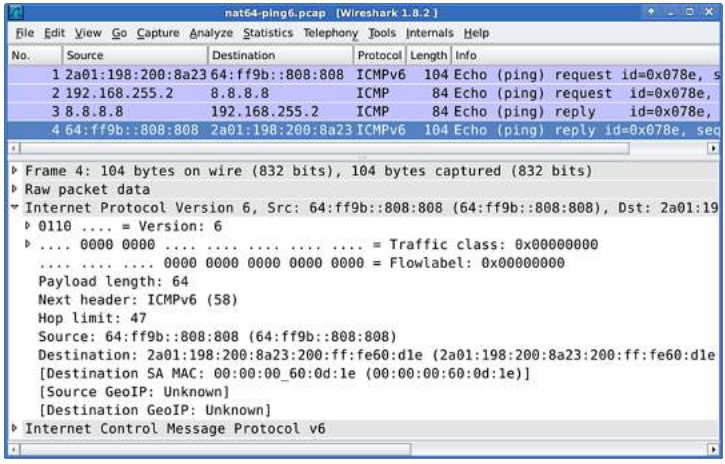


Abbildung 6.19 NAT64-Analyse: Paket 3

Übersetzung von IPv4 nach IPv6

Das IPv4-Datagramm traf beim Empfänger ein und wurde mit einer Echo Reply Message beantwortet. Auch die Antwort wurde von Tayga übersetzt, diesmal von IPv4 nach IPv6. Die IP-Header der Pakete sind in den Abbildungen 6.19 und 6.20 dargestellt. Werfen wir nun einen Blick auf die erwähnten Header. Der wichtigste Unterschied ist wieder die Versionsnummer zu Anfang der Header, denn sie bestimmt den weiteren Aufbau der Header. Die Felder Differentiated Services (IPv4) und Traffic Class (IPv6) enthalten erneut ausschließlich Nullen. Das Flow Label wurde von Tayga ebenfalls auf Null gesetzt, da es kein entsprechendes Feld im IPv4-Header gibt, welches als Informationsquelle dienen könnte. Von der Total Length im IPv4-Header wird die Header Length subtrahiert um die Payload Length für den IPv6-Header zu ermitteln. Eine Fragmentierung hat nicht stattgefunden, darum verwirft Tayga diese Informationen aus dem IPv4-Header ebenso, wie die Header Checksum. Aus der Time To Live (IPv4) hat Tayga das Hop Limit (IPv6) geformt, nicht ohne dabei den Wert zu dekrementieren. Dem Feld Protocol (IPv4) konnte Tayga das Protokoll der Payload entnehmen und hat daraus den Next Header (IPv6) gebildet. Tayga führt eine Tabelle, in der jeder IPv6-Adresse welche NAT64 nutzt, eine eigene IPv4-Adresse aus dem Adresspool zugewiesen ist. Anhand dieser Tabelle

Abbildung 6.20 NAT64-Analyse: Paket 4



können die IPv4-Adressen im IPv4-Header zu IPv6-Adressen für den IPv6-Header umgerechnet werden. So wurde aus der Quelladresse `8.8.8.8` wieder `64:ff9b::808:808`, und aus der Zieladresse `192.168.255.2` wurde die IPv6-Adresse von *lynx*.

6.5 | DNS64 am Link

Problembeschreibung

Die Nutzung von NAT64 alleine bietet leider nicht den gewünschten Komfort am Link *internal*. Ein kleiner Test zeigt die Defizite auf. Mit einem Browser soll auf *lynx* oder *felis* die Website *ipv4.ipv6-workshop.de* aufgerufen werden. Der Vorgang schlägt fehl und wird mit einer Fehlermeldung quittiert, die zum Beispiel *Cannot resolve hostname* lauten könnte. Eigentlich sollte sich hinter dem oben genannten Hostnamen eine IP-Adresse verbergen. Wir überprüfen die Namensauflösung manuell auf *lynx*:

```
user@lynx:~$ host ipv4.ipv6-workshop.de
ipv4.ipv6-workshop.de has address 193.160.39.130
```

Es zeigt sich, dass der Hostname zwar aufgelöst werden konnte, in der Antwort aber keine IPv6-Adresse enthalten war. Damit ist der Host für *lynx* nicht erreichbar.

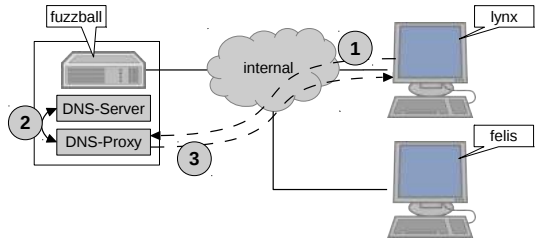


Abbildung 6.21 DNS-Proxy auf *fuzzball* vorschalten

Lösung

Lösen lässt sich das Problem der fehlenden IPv6-Adresse in DNS-Antworten mit DNS64. DNS-Anfragen der Hosts auf dem Link werden bereits jetzt von *fuzzball* zentral beantwortet. Momentan liefert Unbound als zuständiger Resolving DNS Server die Antworten unverändert aus, selbst wenn sie keine IPv6-Adresse enthalten sollten. Das werden wir durch Vorschaltung eines DNS-Proxys ändern. Der Aufbau ist in Abbildung 6.21 zu sehen. Wenn ein Host, zum Beispiel *lynx*, eine DNS-Anfrage zwecks Auflösung an *fuzzball* richtet, dann wird sie vom DNS-Proxy entgegengenommen. Der DNS-Proxy reicht die Anfrage an den Nameserver, auf *fuzzball* ist das Unbound, weiter. Vom Nameserver erhält der Proxy auch die Antwort, die er sogleich auswertet. Findet der DNS-Proxy in der Antwort einen Eintrag für eine IPv4-Adresse aber keinen Eintrag für eine IPv6-Adresse, dann generiert er einen neuen Eintrag. Dieser verweist auf eine IPv6-Adresse, welche aus der IPv4-Adresse und dem NAT64-Präfix erstellt wird. So zwingen wir unsere Hosts am Link dazu, für Hosts ohne IPv6 automatisch NAT64 zu nutzen. Beim normalen Surfen soll auf den Hosts später kein Unterschied zu bemerken sein.

Totd

Als DNS-Proxy werden wir den *Trick or Treat Daemon*, kurz *Totd*, einsetzen.[5] Totd ist klein, schnell und obendrein äußerst einfach zu konfigurieren. Eine Alternative zu der Kombination aus Unbound und Totd ist *Bind9*. Der mächtige Nameserver Bind9 kann neben der Namensauflösung auch NAT64-Adressen generieren.

Notwendige Änderungen

Totd wird auf *fuzzball* auf Port 53 lauschen und dort DNS-Anfragen annehmen. Zur Zeit wird der Port noch von Unbound

[5] *http://www.dillema.net/software/totd.html*

belegt, der daher auf einen anderen Port umziehen muss. Totd wiederrum wird eingehende Anfragen an Unbound durchreichen. Dafür muss Totd die Anfragen an Unbounds neuen Port richten.

Totd installieren

Wir beginnen die Umstellung auf DNS64 mit der Installation von Totd auf *fuzzball*:

```
root@fuzzball:~# apt-get install totd
✂ The following NEW packages will be installed:
    totd
  0 upgraded, 1 newly installed, 0 to remove and 0 not ⤸
      upgraded.
✂ Setting up totd (1.5.1-1.1) ...
```

Totd in bestehenden DNS-Infrastrukturen zu installieren kann problematisch sein. Es ist daher nicht ungewöhnlich wenn bei der Installation Fehler im Zusammenhang mit dem Programm *resolvconf* auftreten. Das Wichtigste ist, dass Totd installiert wurde. Erkennbar ist dies an einer Zeile die mit *Setting up totd* beginnt. Das Fehlschlagen der automatischen Konfiguration ist nicht tragisch, da wir eine manuelle Konfiguration bevorzugen. Um einen Fehler im Totd-Softwarepaket, sowie einen unliebsamen Konfigurationsautomatismus zu umschiffen, müssen wir uns einer Datei entledigen. Damit gewinnen wir zwar keinen Schönheitspreis, ersparen uns aber eine Menge Ärger beim Betrieb von Totd auf Ubuntu Server 12.04:

```
root@fuzzball:~# rm /etc/resolvconf/update.d/totd
```

Unbound umziehen

Nach der Installation von Totd muss zunächst Unbound auf einen anderen Port umziehen. Als neuen Port werden wir 5300 verwenden. Gleichzeitig braucht Unbound auch nicht mehr auf anderen Adressen als der Loopback Address lauschen. Alle Anfragen kommen in Zukunft vom DNS-Proxy, und dieser wird angewiesen die Loopback Address dafür zu benutzen. Die geänderte Konfigurationsdatei zeigt Abbildung 6.22.

Konfiguration von Totd

Totd erwartet seine Konfiguration in der Datei `/etc/totd.conf`. Zum Betrieb reichen wenige Angaben aus. Als erstes muss der DNS-Proxy natürlich wissen, an wen er die Anfragen

Datei: /etc/unbound/unbound.conf

```
server:
        verbosity: 1

        interface: ::1

        port: 5300

        access-control: ::1/128 allow
```

Abbildung 6.22 Konfiguration von Unbound

Datei: /etc/totd.conf

```
forwarder ::1 port 5300

prefix 64:ff9b::

port 53
```

Abbildung 6.23 Konfiguration von Totd

weiterleiten soll. Diese Angaben können wir der Konfigurationsdatei von Unbound entnehmen. Anschließend wird eine Angabe zum verwenden NAT64-Präfix benötigt. Da wir im Workshop das Well-known-Präfix verwenden, tragen wir dieses dort ein. Zuletzt legen wir noch fest, auf welchen Port Totd selbst auf eingehende DNS-Anfragen warten soll. Der freigewordene Standardport für DNS, Port 53, wird es werden. Die komplette Konfigurationsdatei ist in Abbildung 6.23 zu sehen.

Totd starten

Zum Abschluss der DNS64-Installation starten wir Totd:

```
root@fuzzball:~# service totd start
Starting totd: totd
```

DNS64 testen

Wie jede Änderung am Netz, testen wir auch diese. Zuerst auf *fuzzball*:

```
user@fuzzball:~$ host ipv4.ipv6-workshop.de
ipv4.ipv6-workshop.de has address 193.160.39.130
ipv4.ipv6-workshop.de has IPv6 address 64:ff9b::c1a0:2782
```

Der Hostname, für den wir vor kurzem noch keinen IPv6-Eintrag zurückgeliefert bekamen, hat nun eine IPv6-Adresse vom DNS-Proxy zugeteilt bekommen. Ob *fuzzball*, ein Node der selbst mit IPv4-Konnektivität ausgestattet ist, NAT64 benutzen sollte, wäre zu diskutieren. Auf *fuzzball* würde es

ausreichen, weiterhin alleine Unbound zu befragen, ohne den DNS-Proxy zu behelligen.

Viel wichtiger ist aber die Frage, ob die Hosts am Link *internal* die modifizierten Antworten erhalten. Sie haben nämlich keine IPv4-Konnektivität und sind auf ein funktionierendes NAT64 und DNS64 angewiesen, wenn sie Kontakt zum IPv4-Internet aufnehmen müssen. Wir fangen bei *lynx* an:

```
user@lynx:~$ host ipv4.ipv6-workshop.de
ipv4.ipv6-workshop.de has address 193.160.39.130
ipv4.ipv6-workshop.de has IPv6 address 64:ff9b::c1a0:2782
```

Das Ergebnis ist zufriedenstellend. Der Vollständigkeit halber soll auch *felis* diesem Test unterzogen werden:

```
C:\Users\user>nslookup ipv4.ipv6-workshop.de
Name:       ipv4.ipv6-workshop.de
Addresses:  64:ff9b::c1a0:2782
            193.160.39.130
```

NAT64 und DNS64 benutzen

Nutzen Sie die Hosts *lynx* und *felis*, um ein wenig im Internet zu surfen. Finden Sie Websites, die mit NAT64 und DNS64 nicht korrekt funktionieren? Falls ja, nehmen Sie sich Zeit und versuchen Sie die Ursache herauszufinden! Die Methoden und Werkzeuge für eine Fehlersuche in IPv6-Netzwerken haben wir im Workshop gelernt und schon häufig angewandt.

7 | Der Paketfilter

Bis jetzt haben wir uns größtenteils ungeschützt im IPv6-Internet umherbewegt. Auf *felis* finden wir mit der Windows-Firewall einen akzeptablen Grundschutz vor. Der Host *lynx* und der Router *fuzzball* haben aber noch keinen Paketfilter. Mit der Einrichtung von angepassten Paketfiltern werden wir diese Missstände nun beheben. Angepasst deshalb, weil sich die Anforderungen an die Paketfilter von Hosts und Routern unterscheiden. Zunächst schauen wir uns an, wie das Filtern von Paketen unter Linux grundsätzlich funktioniert. Anschließend schreiben wir einen Paketfilter für *lynx*, dabei erlernen wir nebenbei den Umgang mit den wichtigsten Werkzeugen der Filterwartung. Danach schreiben wir einen Paketfilter für *fuzzball*, der dem besonderen Umstand Rechnung trägt, dass *fuzzball* ein Router ist.

7.1 | Einführung in Netfilter

Architektur

Als Paketfilter werden wir das *Netfilter*-Framework des Linux Kernels verwenden. Technisch gesehen befindet sich der Paketfilter damit im sogenannten *Kernelspace*, was erhebliche Geschwindigkeitsvorteile mit sich bringt. Gesteuert wird Netfilter von Kommandos aus dem *Userspace*, zum Beispiel aus einem Terminal heraus.[1] Im Kernelspace, also dort wo die Pakete über Interfaces das System betreten, befinden sich auch

[1] Als Kernelspace wird der Teil des Betriebssystems bezeichnet, in dem der Kernel, und damit auch die Treiber, laufen. Er ist aus dem Userspace, wo die Programme der Nutzer laufen, nicht ohne Umwege erreichbar.

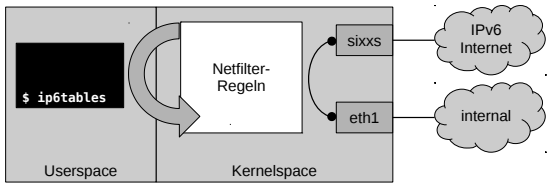


Abbildung 7.1 Netfilter im Kernelspace, ip6tables im Userspace

die Regeln des Paketfilters. Sie können deshalb direkt angewendet werden.

ip6tables Eines der Kommandos zum Verwalten und Anzeigen der Netfilter-Regeln ist `ip6tables`. Es lädt die Netfilter-Regeln aus dem Kernel herunter, modifiziert sie entsprechend den mitgegebenen Parametern, und lädt sie dann wieder in den Kernel hinein. Der Ablauf ist in Abbildung 7.1 dargestellt.

Regeln Wenn man Pakete filtert, möchte man bestimmte Gruppen von Paketen erkennen, um ihnen eine definierte Behandlung zukommen zu lassen. Und sei es nur, um sie wegen ihrer Gefährlichkeit oder Unerwünschtheit zu verwerfen. Derartige Pakete erkennt Netfilter anhand von *Regeln*. In einer Regeln sind die Kriterien festgelegt, denen ein Paket genügen muss, um von der Regel erfasst zu werden.

Chains Gruppen von Regeln organisiert Netfilter in Regelketten, den sogenannten *Chains*. Dabei gibt es eingebaute Chains, beispielsweise *INPUT*, *FORWARD* und *OUTPUT*, aber auch nutzerdefinierte Chains.

Tabellen Die Chains wiederum werden in *Tabellen* zusammengefasst, daher auch der Name *ip6tables* des entsprechenden Werkzeugs. Sofern nicht anders angegeben, nutzen wir die Tabelle *filter*. Bei der Nutzung von `ip6tables` wird diese Tabelle automatisch gewählt, darum verzichten wir bei der Definition von Regeln oft auf die explizite Angabe des Tabellennamens.

Chains der filter-Tabelle Pakete die durch die filter-Tabelle laufen, passieren eine der bereits vordefinierten Chains. Das betrifft sowohl Pakete für die das System nur ein Zwischenstopp ist, als auch Pakete

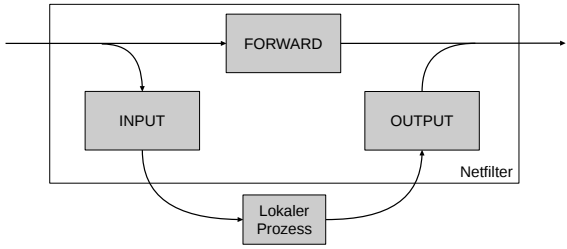


Abbildung 7.2 Die Netfilter-Chains der Tabelle *filter*

die an das System selbst gerichtet sind oder von dort stammen. Pakete für das System landen dann üblicherweise in einem lokalen Prozess oder einem Socket. Darunter können wir uns einen Dienst vorstellen, zum Beispiel einen DNS- oder Webserver. Die eingebauten Chains der filter-Tabelle sind:

INPUT
: Pakete die für einen lokalen Prozess oder Socket bestimmt sind, gehen durch diese Chain.

OUTPUT
: Pakete die von einem lokalen Prozess oder Socket kommen, werden durch diese Chain geschickt.

FORWARD
: Pakete die auf einem Interface eintreffen, und das System sogleich wieder verlassen, passieren diese Chain. Das trifft auch auf Pakete zu, die das System auf dem gleichen Interface verlassen, auf dem sie gekommen sind.

Die Chains der filter-Tabelle sind in auch Abbildung 7.2 dargestellt.

Die filter-Tabelle wird nur dazu genutzt, Pakete zu erkennen und über ihr Schicksal zu entscheiden. In der filter-Tabelle können Pakete nicht verändert werden.

Chains der mangle-Tabelle

Deshalb sei hier auch die Tabelle *mangle* erwähnt. Sie erlaubt nämlich die Modifikation von Paketen. Damit stellt sie ein sehr mächtiges Werkzeug im Netfilter-Framework dar. Unter IPv4 nutzt man die mangle-Tabelle zur Implementierung von SNAT und PAT.

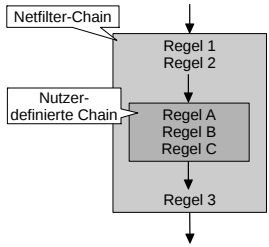


Abbildung 7.3 Nutzerdefinierte Chain innerhalb einer Netfilter-Chain

Zusätzlich zur INPUT-, OUTPUT- und FORWARD-Chain enthält die mangle-Tabelle auch folgende Chains:

PREROUTING
: Hier gehen alle Pakete durch, die über ein Interface eingetroffen sind. Diese können in der Chain verändert werden, noch bevor die Routingentscheidung getroffen wurde.

POSTROUTING
: Kurz bevor Pakete das System verlassen, werden sie durch diese Chain geschickt. Auch hier ist wieder eine Veränderung möglich. Möchte man SNAT nutzen, wäre dies der richtige Ort um die Quelladresse zu manipulieren.

Beispiel

Abbildung 7.3 zeigt die Regeln 1 bis 3 einer Netfilter-Chain und die Regeln A bis C einer nutzerdefinierten Chain.

Die Position jeder einzelnen Regel oder Chain ist entscheidend für die Leistungsfähigkeit und die Sicherheit des Paketfilters. Regeln werden stets in der Reihenfolge ihres Auftretens in einer Tabelle abgearbeitet. Die nutzerdefinierte Chain aus der Abbildung 7.3 ist zwischen den Regeln 2 und 3 der Netfilter-Chain positioniert, somit lautet die endgültige Reihenfolge der Regeln:

- Regel 1
- Regel 2
- Regel A
- Regel B
- Regel C
- Regel 3

First Match

Zu filternde Pakete werden gegen die Regeln geprüft, bis eine der Regeln zutrifft. Hat eine Regel dem Paket ein abschließendes Schicksal bescheinigt, werden die folgenden Regeln nicht mehr durchlaufen. Das Verfahren wird auch *First Match* genannt. Durch die geschickte Positionierung der Regeln lässt sich ein effizienter Paketfilter erstellen.

Default Policies

Nachdem ein Paket eine Chain der filter-Tabelle durchlaufen hat, sollte sein weiteres Schicksal feststehen. Nun kann es vorkommen, dass keine der Regeln in der Chain auf das Paket angewendet werden konnte. Auch besteht nicht die Verpflichtung, überhaupt Regeln in eine Chain einzupflegen. Und so tritt gelegentlich der Fall ein, dass die sogenannte *Default Policy* der Chain greifen muss. Die Default Policy einer Chain legt fest, wie mit Paketen zu verfahren ist, die am Ende der Chain angekommen sind.

Die typischen Default Policies sind:

ACCEPT
: Das Paket wird zugelassen.

DROP
: Das Paket wird verworfen.

Targets

Neben den Kriterien, nach denen ein Paket erkannt wird, definieren die Regeln auch, was bei einem Treffer passieren soll. Im Netfilter-Jargon spricht man von einem *Target*, und meint damit das weitere Schicksal des betroffenen Paketes. Zwei der eingebauten Targets, und zwar *ACCEPT* und *DROP*, haben wir bei den Default Policies schon kennengelernt. Es existieren noch weitere Targets:

RETURN
: Die aktuelle Chain wird verlassen. Von einer nutzerdefinierte Chain springt man hiermit zurück in die Netfilter-Chain. Befindet sich die Regel, welche dieses Target nutzt, bereits in einer Netfilter-Chain, dann wird die Default Policy angewendet.

REJECT
: Das Paket wird verworfen, die Quelle erhält eine Fehlermeldung. Die Fehlermeldung wird als ICMPv6-Nachricht

formuliert. Eine Ausnahme kann bei TCP-Paketen konfiguriert werden, dort ist auch ein Verbindungsabbruch mit TCP-Reset einstellbar.

LOG

Der Kernel schreibt eine Zeile, mit den wichtigsten Header-Informationen des Paketes, in das Systemlog. Dies ist ein besonderes Target, denn es beendet das Filtern nicht. Der Filter fährt bei der nächsten Regel fort.

TOS

Das Feld Traffic Class im IPv6-Header wird mit einem zu definierenden Wert überschrieben. Da hier eine Veränderung des Paketes stattfindet, ist dieses Target nur in der mangle-Tabelle erlaubt.

QUEUE, NFQUEUE

Das Paket wird in den Userspace befördert, wo es von einem Prozess verarbeitet werden kann. Das Target wird genutzt um aufwändige Analysen durchzuführen, die im Kernelspace wegen ihrer Komplexität nicht gut aufgehoben wären.

Nutzerdefinierte Chains als Targets

Alternativ kann als Target auch eine nutzerdefinierte Chain angegeben werden. Die dadurch gewonnene Flexibilität ist enorm. So kann man verdächtige Pakete durch eine Chain mit besonders strengen Regeln schicken, oder vertraute Adressbereiche an langsamen Chains vorbeischleusen. Wir werden diese Funktion später nutzen, um potentiell gefährliche Protokolle in speziellen Chains zu entschärfen.

Connection Tracking

Das Netfilter-Framework unterstützt *Connection Tracking*, ein Feature mit dem sich *Stateful Packet Inspection* (SPI) realisieren lässt. Das heißt, wir können unterschiedliche Regeln auf Pakete anwenden, abhängig davon, ob sie eine neue oder eine bestehende Verbindung repräsentieren. Damit wird ein Kompromiss möglich, der es erlaubt neue Verbindungen mit aufwändigen Regeln zu bearbeiten. Trotzdem werden bestehende Verbindungen nicht negativ beeinflusst, da wir die entsprechenden Pakete zügig passieren lassen können. Über jede Verbindung wird vom Connection Tracking anhand eines Datensatzes Buch geführt. Dieser besteht aus dem Netzwerkprotokoll (IPv6), der Quell- und Zieladresse, dem Upper

Layer Protocol und zusätzlichen Daten des Upper Layer Protocol.

Connection Tracking mit TCP und UDP

Die zusätzlichen Daten des Upper Layer Protocols sind bei TCP die Portnummern von Quelle und Ziel. Gleichzeitig bietet die TCP-Statemachine Netfilter weitere Anhaltspunkte für das Connection Tracking.[2] Bei UDP stehen nur die Portnummern zur Verfügung, da es sich um ein verbindungsloses Protokoll handelt.

Connection Tracking mit anderen Protokollen

Wenn eine Verbindung keine zusätzlichen Daten bereitstellt, wird eine Pseudo-Verbindung angenommen. Sie basiert auf den wenigen vorhandenen Daten des Datensatzes der Verbindung. Dann arbeitet die Plausibilitätsprüfung für solche Verbindungen mit vordefinierten Ablaufzeiten für Pakete. Bleiben passende Pakete während eines gewissen Zeitraums aus, gilt die Verbindung als beendet.

Verwandte Pakete

Das Connection Tracking des Netfilter-Frameworks unterscheidet zwischen zugehörigen und verwandten Paketen. Pakete die einer Verbindung zugehörig sind, dienen dem Austausch von Daten in der Verbindung. Es handelt sich um jene Pakete, die zwischen den beteiligten Nodes hin und her wandern. Verwandte Pakete hingegen sind jene Pakete, die zwar einer Verbindung zugeordnet werden können, dieser aber nicht direkt angehören. Damit sind hauptsächlich ICMPv6-Fehlermeldungen gemeint, ohne die eine Verbindung häufig nicht zustande kommen kann. Verwandte Pakete sind daher ebenso wichtig für eine Verbindung wie die zugehörigen Pakete.

Zustände im Connection Tracking

Die verschiedenen Zustände einer Verbindung lassen sich bequem in den Netfilter-Regeln abfragen. Ein Paket kann also nicht nur nach seiner Herkunft, seinen Eigenschaften und seinen Header-Feldern bewertet werden, sondern auch danach, in welchem Zusammenhang es zu bereits behandelten Paketen steht. Die wichtigsten Zustände lauten:

[2] Die TCP-Statemachine liefert detaillierte Informationen über den aktuellen Zustand einer TCP-Verbindung. Gerade bei Verbindungen, welche noch nicht vollständig ausgehandelt sind, erleichtern diese Informationen die Bewertung der Plausibilität eintreffender Pakete.

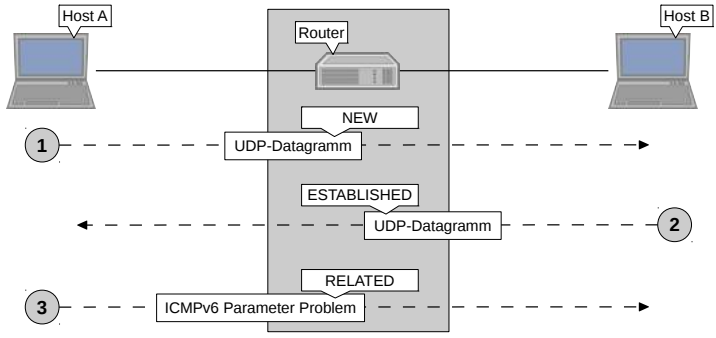


Abbildung 7.4 Beispiel für die Zustände des Connection Trackings

NEW
: Mit diesem Paket würde eine neue Verbindung im Connection Tracking geführt werden. Es repräsentiert die erste Kontaktaufnahme durch, oder mit, einem anderen Node.

ESTABLISHED
: Das Paket gehört zu einer bestehenden Verbindung.

RELATED
: Das Paket ist mit einer bestehenden Verbindung verwandt und für das Funktionieren dieser wahrscheinlich wichtig. Das Auftreten des Paketes wurde vom Connection Tracking bereits erwartet.

INVALID
: Das Paket ist ungültig. Es wurde entweder nicht erwartet, oder es passt nicht zu der Verbindung, in der es auftrat. Falsch gesetzte Flags in TCP-Segmenten führen ebenfalls zu diesem Zustand.

UNTRACKED
: Das Paket wird vom Connection Tracking ignoriert. Diesen Zustand muss der Administrator des Paketfilters manuell vergeben, es findet nur in Sonderfällen oder bei der Fehlersuche Anwendung.

Beispiel Connection Tracking

Das Beispiel aus Abbildung 7.4 verdeutlicht die Zustände des Connection Trackings. Host A und Host B kommunizieren über den gemeinsamen Router. Auf dem Router läuft ein Linux-Kernel mit Netfilter. Das erste Paket, das der Router sieht, ist ein UDP-Datagramm von Host A an Host B. Da das Connection Tracking auf dem Router dieses Paket keiner bestehenden

Verbindung zuordnen kann, erstellt er eine neue Pseudo-Verbindung. Das Paket hat den Zustand NEW. Kurz darauf trifft ein Antwortpaket von Host B ein. Das Connection Tracking kann das Paket der eben erstellten Verbindung zuordnen und versieht das Paket daher mit dem Zustand ESTABLISHED. Leider konnte der Host A das Paket nicht verarbeiten, der Grund könnte zum Beispiel ein unbekannter Parameter in einem Extension Header gewesen sein. Der Host A antwortet mit der ICMPv6-Fehlermeldung Parameter Problem. Auf dem Router erkennt das Connection Tracking, dass es sich hierbei um ein verwandtes Paket handelt und ordnet es der bestehenden Verbindung zu. Es erhält den Zustand RELATED.

7.2 | Host-Paketfilter

Nach der Theorie zu Netfilter werden wir uns nun wieder der Praxis widmen. Mit dem Kommando `ip6tables` können wir, wir erinnern uns, die Netfilter-Regeln verändern.

ip6tables benutzen

Genau das werden wir jetzt auf *lynx* ausprobieren. Wir schreiben uns einen Paketfilter für *lynx*. Nicht nur um uns mit den Parametern von `ip6tables` vertraut zu machen, sondern auch um *lynx* zu schützen. Dabei sind im Workshop mehrere Varianten des gleichen Kommandos angegeben, eine Lang- und eine Kurzform. Es reicht natürlich, nur eine der beiden Varianten einzugeben. Die Kürzere ist schneller getippt, die Längere hilft gerade Anfängern zu verstehen, welche Bedeutung die Parameter haben.

Regeln in Netfilter-Chains löschen

Der erste Schritt beim Erstellen eines neuen Paketfilters ist das Beseitigen von Altlasten. Alle Regeln der Netfilter-Chains müssen gelöscht werden, damit sie nicht mit den neuen Regeln in Konflikt treten. Im Netfilter-Jargon spricht man vom sogenannten *flushen* der Chains. Für jede der relevanten Tabellen muss ein eigener Flush durchgeführt werden.

Die Kommandos lauten:

```
root@lynx:~# ip6tables --flush --table filter
root@lynx:~# ip6tables --flush --table mangle
```

Alternative Kurzformen:

```
root@lynx:~# ip6tables -F
root@lynx:~# ip6tables -F -t mangle
```

Sowohl in der Lang- als auch in der Kurzform können wir uns die Angabe der Tabelle filter sparen. Bei fehlender Angabe geht `ip6tables` immer davon aus, dass wir die Tabelle filter meinen.

Alle nutzerdefinierten Chains löschen

Wir haben die Tabellen erfolgreich von allen Regeln befreit. Was jetzt noch fehlt, ist das Löschen von nutzerdefinierten Chains, die vielleicht noch in den Netfilter-Chains der Tabellen eingehängt sind.

Die Kommandos lauten:

```
root@lynx:~# ip6tables --delete-chain --table filter
root@lynx:~# ip6tables --delete-chain --table mangle
```

Alternative Kurzformen:

```
root@lynx:~# ip6tables -X
root@lynx:~# ip6tables -X -t mangle
```

Regeln auflisten

Vom Erfolg unserer Löschaktionen können wir uns überzeugen, indem wir `ip6tables` mit dem Parameter zur Auflistung der Regeln aufrufen. Es sollten keine Regeln angezeigt werden. Das Kommando lautet:

```
root@lynx:~# ip6tables --list
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

Alternative Kurzform:

```
root@lynx:~# ip6tables -L
```

Wir sehen die Chains INPUT, FORWARD und OUTPUT ohne Regeln. Die Spalten *target*, *prot* (Protokoll), *opt* (Optionen), *source* und *destination* sind jeweils leer.

Default Policy setzen

Eine Sache jedoch fällt auf: Die Default Policies sind alle auf das Target ACCEPT eingestellt. Damit würde der Paketfilter alle Pakete, die nicht von einer Regel betroffen sind, akzeptieren. Das Verfahren wird auch Blacklisting genannt, da man explizite Regeln für unerwünschte Pakete definieren muss. Für die Chains INPUT und FORWARD werden wir aber Whitelisting verwenden. Das heißt, eingehende Pakete müssen von einer unserer Regeln erfasst und für unschädlich befunden werden. Andernfalls sollen sie verworfen werden. Die Default Policies der betroffen Chains werden wir jetzt auf das Target DROP einstellen.

Die Kommandos lauten:

```
root@lynx:~# ip6tables --policy INPUT DROP
root@lynx:~# ip6tables --policy FORWARD DROP
```

Alternative Kurzformen:

```
root@lynx:~# ip6tables -P INPUT DROP
root@lynx:~# ip6tables -P FORWARD DROP
```

Wir listen die Regeln wieder auf, um uns vom Erfolg der Maßnahme zu überzeugen:

```
root@lynx:~# ip6tables --list
Chain INPUT (policy DROP)
target     prot opt source               destination

Chain FORWARD (policy DROP)
target     prot opt source               destination
```

Regel hinzufügen, Sprünge definieren

Wenn man einen Paketfilter, so wie wir, Schritt für Schritt aufbaut, dann möchte man die Regeln nach und nach hinten anfügen. Bei `ip6tables` wird dann von *append* gesprochen. Eine Regel für ein Paket soll nicht nur das Paket erfassen, sondern auch eine Entscheidung über sein weiteres Schicksal treffen. Diese Entscheidung wird mit einem Sprung zu einem Target realisiert. Da wir auf *lynx* keine Pakete weiterleiten werden, es handelt sich schließlich nicht um einen Router, fügen wir eine

Regeln an die FORWARD-Chain an, die alle Pakete verwirft. Das Verwerfen geschieht durch einen Sprung in das DROP-Target.

Das Kommando lautet:

```
root@lynx:~# ip6tables --append FORWARD --jump DROP
```

Alternative Kurzform:

```
root@lynx:~# ip6tables -A FORWARD -j DROP
```

Ein kurzer Blick auf die FORWARD-Chain zeigt, dass die Regel wie erwartet angefügt wurde:

```
root@lynx:~# ip6tables --list FORWARD
Chain FORWARD (policy DROP)
target     prot opt source               destination
DROP       all      anywhere             anywhere
```

Übrigens, diese Regel wäre natürlich nicht nötig gewesen, denn die Default Policy der FORWARD-Chain hätte schon für einen Sprung ins DROP-Target gesorgt. Wir haben diese Regel nur zu Übungszwecken eingefügt.

Regeln löschen

Belasten wir Netfilter lieber nicht mit unnötigen Regeln, und löschen die eben angefügte Regel aus der FORWARD-Chain wieder. Das Löschen von Regeln verläuft ähnlich dem hinzufügen. Die Regel wird mit allen ihren Parametern übergeben, auch mit dem ursprünglichen Sprungparameter. Entscheidend ist der Aufruf des Parameters *delete* anstatt *append*.

Das Kommando lautet:

```
root@lynx:~# ip6tables --delete FORWARD --jump DROP
```

Alternative Kurzform:

```
root@lynx:~# ip6tables -D FORWARD -j DROP
```

Ein erneuter Blick auf die FORWARD-Chain zeigt, dass die Regel entfernt wurde:

```
root@lynx:~# ip6tables --list FORWARD
Chain FORWARD (policy DROP)
target     prot opt source               destination
```

Im Moment erlauben wir keine eingehenden Pakete, auch nicht jene vom Loopback Interface. Ein kleiner Test zeigt diesen offensichtlichen Missstand auf:

Parameter Interface

```
user@lynx:~$ ping6 -c 3 ::1
PING ::1 (::1) 56 data bytes

--- ::1 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, ⤸
     time 2008ms
```

Datenverkehr auf dem Loopback Interface verlässt den Node per Definition nie. Es gibt eigentlich keine Gründe, wieso wir ihn verbieten sollten. Daher erlauben wir alle Pakete, sofern sie über das Loopback-Interface hinein oder hinaus wollen. Die Parameter unterscheiden sich je nach betroffener Chain, in der INPUT-Chain benutzen wir den Parameter für das Eingangs-Interface, in der OUTPUT-Chain den für das Ausgangs-Interface. In der FORWARD-Chain könnten wir, wenn wir wollten, sogar beide Parameter gleichzeitig benutzen.

Die Kommandos lauten:

```
root@lynx:~# ip6tables --append INPUT --in-interface lo ⤸
     --jump ACCEPT
root@lynx:~# ip6tables --append OUTPUT --out-interface ⤸
     lo --jump ACCEPT
```

Alternative Kurzformen:

```
root@lynx:~# ip6tables -A INPUT -i lo -j ACCEPT
root@lynx:~# ip6tables -A OUTPUT -o lo -j ACCEPT
```

Jetzt klappt es auch mit dem Echo Request an die Loopback Address:

```
user@lynx:~$ ping6 -c 3 ::1
PING ::1 (::1) 56 data bytes
64 bytes from ::1: icmp_seq=1 ttl=64 time=0.123 ms
3 packets transmitted, 3 received, 0% packet loss, time ⤸
     2000ms
```

Wenn man es ganz genau nimmt, war der Eintrag in der OUTPUT-Chain nicht notwendig, denn diese springt per Default Policy in das Target ACCEPT. Um uns die Möglichkeit offen zu halten, die Default Policy der OUTPUT-Chain noch zu ändern, tragen wir diese Regel dennoch ein. Die Regeln sind so allgemein gehalten, dass sie Netfilter nicht wahrnehmbar belasten.

Nutzerdefinierte Chain erstellen

Ein sehr mächtiges Werkzeug von Netfilter sind die nutzerdefinierten Chains. Zu Übungszwecken werden wir gleich zwei von ihnen erstellen.

Die Kommandos lauten:

```
root@lynx:~# ip6tables --new-chain allwatchedover
root@lynx:~# ip6tables --new-chain lovinggrace
```

Alternative Kurzformen:

```
root@lynx:~# ip6tables -N allwatchedover
root@lynx:~# ip6tables -N lovinggrace
```

Nutzerdefinierte Chain löschen

Für *liebevolle Gunst* ist in der objektiven Welt eines Paketfilters kein Platz, die Chain *lovinggrace* soll daher wieder gelöscht werden.

Das Kommando lautet:

```
root@lynx:~# ip6tables --delete-chain lovinggrace
```

Alternative Kurzform:

```
root@lynx:~# ip6tables -X lovinggrace
```

Nutzerdefinierte Chain umbenennen

Auch der Name der Chain *allwatchedover* ist nicht perfekt. Denn sie soll später dazu dienen, Pakete die in Ungnade gefallen sind, vor dem Verwerfen noch im Systemlog zu notieren. Passender wäre wohl ein Name wie *trashlog*, daher benennen wir die Chain um.

Das Kommando lautet:

```
root@lynx:~# ip6tables --rename-chain allwatchedover ⤸
    trashlog
```

Alternative Kurzform:

```
root@lynx:~# ip6tables -E allwatchedover trashlog
```

Nutzerdefinierte Chain füllen

Die trashlog-Chain wird allerlei unerwünschten Paketen als Target dienen. Ihre wichtigste Aufgabe ist daher das Verwerfen der Pakete. Mit einer Default Policy lässt sich diese Aufgabe nicht erledigen, da nutzerdefinierte Chains keine Default Policies besitzen dürfen. Es bleibt also nur der Sprung zum DROP-Target aus der Chain selbst heraus. Gerade wenn ein Paketfilter noch nicht ausgereift ist, kann es sehr hilfreich sein, mitzuschreiben welche Pakete als unerwünscht erkannt wurden. Häufig übersieht man eine Kleinigkeit und verwirft fälschlicherweise legitime Pakete. Vor dem endgültigen Verwerfen werden wir die Pakete daher, versehen mit einem Hinweis auf die trashlog-Chain, in das Systemlog schreiben lassen. Weil Netfilter Meldungen im Systemlog standardmäßig mit dem Loglevel *warning* versieht, müssen wir auch hier eine Anpassung vornehmen. Warnungen sind für wichtigere Dinge reserviert, es reicht wenn wir von unerwünschten Paketen Notiz nehmen.

Die Kommandos lauten:

```
root@lynx:~# ip6tables --append trashlog --jump LOG ⤸
     --log-level notice --log-prefix "TRASHLOG: "
root@lynx:~# ip6tables --append trashlog --jump DROP
```

Alternative Kurzformen:

```
root@lynx:~# ip6tables -A trashlog -j LOG --log-level ⤸
     notice --log-prefix "TRASHLOG: "
root@lynx:~# ip6tables -A trashlog -j DROP
```

Übrigens, mit *log-level* und *log-prefix* sind uns soeben die ersten Parameter begegnet, von denen es keine Kurzform gibt.

Unsere nutzerdefinierte Chain lassen wir uns wieder von `ip6tables` anzeigen:

```
root@lynx:~# ip6tables --list trashlog
Chain trashlog (0 references)
target     prot opt source               destination
LOG        all      anywhere             anywhere ⤸
               LOG level notice prefix 'TRASHLOG: '
DROP       all      anywhere             anywhere
```

Der Auflistung können wir gleich mehrere Informationen entnehmen. Zum einen sehen wir, das die Chain noch nicht von einer Regel als Target genutzt wird (*0 references*). Zum anderen sehen wir hier auch, mit welchem Loglevel die Meldungen in das Systemlog geschrieben werden, nämlich *notice*. Dem aufmerksamen Betrachter wird auch nicht entgangen sein, dass für diese Chain keine Default Policy angezeigt wird. Denn nutzerdefinierte Chains dürfen bekanntlich keine Default Policy besitzen.

Connection Tracking

Das Connection Tracking wird als Erweiterung, als sogenannte *Match Extension*, nachgeladen. Wenn eine Erweiterung geladen wird, sind zusätzliche Parameter nutzbar, die von der Erweiterung eingebracht werden. Wir werden die Erweiterung Connection Tracking häufig benutzen. Zunächst sollen alle eingehenden Pakete, die vom Connection Tracking als ungültig erkannt wurden, in die nutzerdefinierte Chain *trashlog* springen. Dort werden sie im Systemlog vermerkt und dann verworfen. Das Kommando lautet:

```
root@lynx:~# ip6tables --append INPUT --match conntrack ⤸
    --ctstate INVALID --jump trashlog
```

Alternative Kurzform:

```
root@lynx:~# ip6tables -A INPUT -m conntrack --ctstate ⤸
    INVALID -j trashlog
```

Bestehende Verbindungen und ihre verwandten Pakete sollen den Paketfilter ungehindert passieren können. Daher erlauben wir alle eingehenden Pakete die das Connection Tracking einer bestehenden Verbindung zuordnen konnte.

Das Kommando lautet:

```
root@lynx:~# ip6tables --append INPUT --match conntrack ⤸
    --ctstate ESTABLISHED,RELATED --jump ACCEPT
```

Alternative Kurzform:

```
root@lynx:~# ip6tables -A INPUT -m conntrack --ctstate ⤸
     ESTABLISHED,RELATED -j ACCEPT
```

Parameter Protokoll

Mit dem Parameter *protocol*, in der Kurzform schlicht *p*, lässt sich das Protokoll definieren, auf das eine Regel zutreffen muss. Das Protokoll kann als Protokollnummer, so wie sie im IPv6-Header auftaucht, angegeben werden. Alternativ kann auch die ausgeschriebene Variante genutzt werden. Die gängigsten Protokolle im Überblick:

tcp
: Transmission Control Protocol

udp
: User Datagram Protocol

ipv6
: IPv6-in-IPv6

icmpv6
: Internet Control Message Protocol v6

Wenn in einer Regel das Protokoll spezifiziert wird, dann werden weitere Parameter nutzbar. Bei ICMPv6 wird unter anderem der Parameter *icmpv6-type* nutzbar. Ihm kann die Typnummer einer ICMPv6 Message als Zahl oder ausgeschriebene Variante übergeben werden. Die ausgeschriebenen Typen, manche von ihnen sogar mit alternativer Schreibweise, sind:

- destination-unreachable
- no-route
- communication-prohibited
- address-unreachable
- port-unreachable
- packet-too-big
- time-exceeded, ttl-exceeded
- ttl-zero-during-transit
- ttl-zero-during-reassembly
- parameter-problem
- bad-header
- unknown-header-type

- unknown-option
- echo-request, ping
- echo-reply, pong
- router-solicitation
- router-advertisement
- neighbour-solicitation, neighbor-solicitation
- neighbour-advertisement, neighbor-advertisement
- redirect

NDP und SLAAC erlauben

Wie Sie sicher schon bemerkt haben, hat der NetworkManager auf *lynx* inzwischen seine Konnektivität verloren. Das liegt unter anderem daran, dass ICMPv6 Messages vom Paketfilter noch nicht akzeptiert werden. Wie wir wissen, organisiert sich der Link aber ausschließlich über ICMPv6 Messages. Wir sollten diese also zügig wieder zulassen, um die Konnektivität zurückzugewinnen. Eine Einschränkung werden wir dennoch vornehmen, wir erlauben nur ICMPv6 Messages, die sowohl vom Hop Limit als auch vom Typ her in die typische Kommunikation des Neighbor Discovery Protocols hineinpassen.

Dazu erstellen wir uns eine nutzerdefinierte Chain namens *ndp-slaac*, füllen diese mit Regeln, und hängen sie abschließend für alle ICMPv6-Pakete in die INPUT-Chain ein:

```
root@lynx:~# ip6tables --new-chain ndp-slaac
```

Eine zusätzliche Erweiterung, sie heißt *hl*, ist für die Analyse des Hop Limits zuständig. Wir laden sie wie gewohnt mit *match* nach.

In den ersten Regeln der Chain ndp-slaac erlauben wir NDP:

```
root@lynx:~# ip6tables --append ndp-slaac --protocol ⤶
     icmpv6 --icmpv6-type router-solicitation --match hl ⤶
     --hl-eq 255 --jump ACCEPT
root@lynx:~# ip6tables --append ndp-slaac --protocol ⤶
     icmpv6 --icmpv6-type router-advertisement --match ⤶
     hl --hl-eq 255 --jump ACCEPT
root@lynx:~# ip6tables --append ndp-slaac --protocol ⤶
     icmpv6 --icmpv6-type neighbor-solicitation --match ⤶
     hl --hl-eq 255 --jump ACCEPT
root@lynx:~# ip6tables --append ndp-slaac --protocol ⤶
     icmpv6 --icmpv6-type neighbor-advertisement --match ⤶
     hl --hl-eq 255 --jump ACCEPT
```

Auch Umleitungen werden wir akzeptieren:

```
root@lynx:~# ip6tables --append ndp-slaac --protocol ⤶
     icmpv6 --icmpv6-type redirect --match hl --hl-eq ⤶
     255 --jump ACCEPT
```

Und natürlich MLDv1 und MLDv2, um die Selbstorganisation des Links nicht zu gefährden:

```
# Typ 130: Multicast Listener Query (MLDv1, MLDv2)
root@lynx:~# ip6tables --append ndp-slaac --protocol ⤶
     icmpv6 --icmpv6-type 130 --match hl --hl-eq 1 ⤶
     --jump ACCEPT
# Typ 131: Multicast Listener Report (MLDv1)
root@lynx:~# ip6tables --append ndp-slaac --protocol ⤶
     icmpv6 --icmpv6-type 131 --match hl --hl-eq 1 ⤶
     --jump ACCEPT
# Typ 132: Multicast Listener Done (MLDv1)
root@lynx:~# ip6tables --append ndp-slaac --protocol ⤶
     icmpv6 --icmpv6-type 132 --match hl --hl-eq 1 ⤶
     --jump ACCEPT
# Typ 143: Multicast Listener Report (MLDv2)
root@lynx:~# ip6tables --append ndp-slaac --protocol ⤶
     icmpv6 --icmpv6-type 143 --match hl --hl-eq 1 ⤶
     --jump ACCEPT
```

Zuletzt wird die nutzerdefinierte Chain noch in die INPUT-Chain eingehängt:

```
root@lynx:~# ip6tables --append INPUT --protocol icmpv6 ⤶
     --jump ndp-slaac
```

Nun sollte der NetworkManager wieder in der Lage sein, Konnektivität herzustellen.

Aus Platzgründen wird hier auf eine Angabe der alternativen Kurzform verzichtet.

Parameter Quelle und Ziel

Die wohl wichtigsten Parameter eines jeden Paketfilters sind Quell- und Zieladressen von Paketen. Dabei ist die Angabe von einzelnen Adressen, von Listen (mit Kommata getrennte Adressen) oder Präfixen möglich. Die Leistungsfähigkeit des Paketfilters hängt stark davon ab, wie gut die Adressen *aggregiert* sind. Das heißt, immer wenn es möglich ist, mehrere Adressen als Präfix auszudrücken, sollte dies geschehen.

Um die Fehlersuche am internen Link zu erleichtern, erlauben wir Echo Request aus dem Präfix für Link-local Addresses. Das Kommando lautet:

```
root@lynx:~# ip6tables --append INPUT --source fe80::/10 ⤸
     --destination fe80::/10 --protocol icmpv6 ⤸
     --icmpv6-type echo-request --match conntrack ⤸
     --ctstate NEW --jump ACCEPT
```

Alternative Kurzform:

```
root@lynx:~# ip6tables -A INPUT -s fe80::/10 -d ⤸
     fe80::/10 -p icmpv6 --icmpv6-type echo-request -m ⤸
     conntrack --ctstate NEW -j ACCEPT
```

Parameter Port

Wenn wir die Regeln auf ein Upper Layer Protocol anwenden, können wir auch auf die diversen Parameter des jeweiligen Protokolls zugreifen. Beispielhaft sei hier der Parameter zum Filtern von Portnummern erwähnt. Auch hier sind wieder mehrere Angaben möglich. Neben alleinstehenden Portnummern sind auch durch Kommata getrennte Listen, aber auch ganze Portbereiche gültige Werte. Portbereiche werden in der Form Erster-Port:Letzter-Port angegeben. Ausgeschriebene Varianten existieren ebenfalls, die komplette Liste befindet sich in der Datei `/etc/services` und wird hier aus Platzgründen ausgespart. Wir nutzen den Parameter Port, um auf *lynx* den Fernwartungsdienst *Secure Shell* (SSH) für den internen Link freizugeben.

Das Kommando lautet:

```
root@lynx:~# ip6tables --append INPUT --source fe80::/10 ⤸
     --protocol tcp --destination-port 22 --match ⤸
     conntrack --ctstate NEW --jump ACCEPT
```

Alternative Kurzform:

```
root@lynx:~# ip6tables -A INPUT -s fe80::/10 -p tcp ⤸
     --dport 22 -m conntrack --ctstate NEW -j ACCEPT
```

Ausgehende Verbindungen

Die ausgehenden Verbindungen von *lynx* werden vom Connection Tracking erfasst. Alle eingehenden Pakete, die zu diesen Verbindungen gehören, werden über die Zustände ESTABLISHED und RELATED ins System hineingelassen. Deshalb sind, sofern man bei dieser großzügigen Konfiguration für ausgehende Verbindungen bleiben möchte, keine weiteren Regeln für die OUTPUT-Chain nötig.

Regeln sichern

Wir haben jetzt einen funktionierenden Paketfilter auf *lynx*, der leider nicht von Dauer ist. Denn die Netfilter-Regeln überleben einen Neustart nicht. Das Kommando `ip6tables-save` kann uns aber weiterhelfen. Es liest alle Regeln aus und gibt sie in einem Format wieder, das für `ip6tables` lesbar ist:

```
root@lynx:~# ip6tables-save
✂
-A INPUT -i lo -j ACCEPT
-A INPUT -m conntrack --ctstate INVALID -j trashlog
-A INPUT -m conntrack --ctstate RELATED,ESTABLISHED -j ⤸
     ACCEPT
-A INPUT -p ipv6-icmp -j ndp-slaac
✂
COMMIT
```

Wir leiten die Ausgabe von `ip6tables-save` in eine Datei um:

```
root@lynx:~# ip6tables-save > /etc/packetfilter
```

So stehen uns die Regeln auch nach einem Neustart noch zur Verfügung, wir müssen sie dann nur noch in den Kernel laden.

Die Datei `/etc/packetfilter` von *lynx* ist in Anhang C *Paketfilter von lynx* zu sehen.

Regeln richtig laden

Es gibt zwei Möglichkeiten, Netfilter nach dem Systemstart wieder mit den vorher definierten Regeln zu versorgen. Die beliebteste scheint zu sein, die mit `ip6tables` beginnenden Kommandos in ein Shell-Skript zu schreiben, und dieses auszuführen. Davon soll an dieser Stelle dringend abgeraten werden! Uns ist aus dem Abschnitt 7.1 *Einführung in Netfilter* bekannt, dass bei jedem Aufruf von `ip6tables` der gesamte Regelsatz aus dem Kernel heruntergeladen wird. Dann wird der Regelsatz modifiziert, zum Beispiel durch anfügen einer Regel. Der geänderte Regelsatz wird dann wieder in den Kernel geladen. Auch wenn der Vorgang blitzartig vonstatten zu gehen scheint, so stellt er doch eine unnötige Belastung für das System dar. Ein weiterer, viel wichtigerer, Grund gegen das eben beschriebene Vorgehen ist im Herzen des Betriebssystems zu finden: Dem Scheduler. Der Scheduler legt fest, wann ein Prozess Rechenzeit bekommt, und wann ihm zugunsten eines anderen Prozesses Rechenzeit entzogen wird. Jeder Aufruf von `ip6tables` ist ein eigener Prozess. Das bedeutet, dass es vorkommen kann, dass der Scheduler einem der vielen `ip6tables`-Prozesse zu irgendeinem Zeitpunkt Rechenzeit entzieht. Andere Prozesse belegen dann den Prozessor bis `ip6tables` diesen dann wieder in Beschlag nehmen darf. Während `ip6tables` pausiert, haben wir einen unfertigen Paketfilter im Kernel, denn es stehen unter Umständen noch Regeln zum Laden aus. Dass ein halbfertiger Paketfilter, auch wenn er nur wenige Millisekunden existiert, keine gute Idee ist, leuchtet ein.

Regeln wiederherstellen

Das Kommando `ip6tables-restore` bietet für das Problem eine Lösung an. Es lädt den gesamten Regelsatz auf einmal in den Kernel, und lässt sich dabei auch nicht unterbrechen. Wir laden unseren Packetfilter testweise erneut in den Kernel:

```
root@lynx:~# ip6tables-restore < /etc/packetfilter
```

Beachten Sie bitte die Richtung in welche die spitze Klammer zeigt. Eine falsch gesetze Klammer wird zu Datenverlust in der Datei `/etc/packetfilter` führen.

Datei: /etc/rc.local

```
1 #!/bin/sh -e
2 #
3 # By default this script does nothing.
4
5 ip6tables-restore < /etc/packetfilter
6
7 exit 0
```

Abbildung 7.5 Skript rc.local

Wiederherstellung nach Systemstart

Auf *lynx* machen wir es uns einfach, indem wir die Wiederherstellung nach dem Systemstart automatisch ausführen. Dazu verändern wir die Datei `/etc/rc.local` wie in Abbildung 7.5 gezeigt.

Faustregel

Als Faustregel gilt: Den Paketfilter mit `ip6tables` erstellen und modifizieren, anschließend mit `ip6tables-save` abspeichern. Den Paketfilter im produktiven Betrieb immer mit `ip6tables-restore` in den Kernel laden.

Weiterführendes Studium

Wir haben soeben `ip6tables` im Schnellverfahren kennengelernt. Die vorgestellten Parameter sind nur ein kleiner Teil des riesigen Funktionsumfangs von *ip6tables* und dem Netfilter-Framework. Bitte nehmen Sie sich etwas Zeit, um im Umgang mit den vorgestellten Werkzeugen Handlungssicherheit zu erlangen. Spätestens wenn wir den komplexeren Paketfilter für *fuzzball* schreiben, wird dies hilfreich sein. Wie so oft hilft ein Studium der Hilfeseiten, abrufbar mit `man ip6tables`, weiter.

7.3 | Router-Paketfilter

Anforderungen

Das Filtern von Paketen auf einem Router stellt eine besondere Herausforderung dar. Auf der einen Seite sollen sich Router, die zwischen zwei kommunizierenden Hosts sitzen, nicht unnötig in die Kommunikation einmischen oder diese gar beeinträchtigen. Auf der anderen Seite möchte man als Betreiber eines Routers nach Möglichkeit keine unerwünschten oder schädlichen Pakete verarbeiten. Darüber hinaus kann ein Router oft nur Vermutungen darüber anstellen, ob ein Paket erwünscht, unerwünscht oder gar schädlich ist. Manchmal lässt

sich nicht einmal feststellen, ob ein Paket in einem gewissen Kontext überhaupt Sinn ergibt. Nicht zuletzt deshalb ist es auch umstritten, ob auf einem Router überhaupt Pakete gefiltert werden sollten. Da *fuzzball* bei uns nicht nur für das reine Routing zuständig ist, sondern auch einen Resolving DNS Server bereitstellt und als NAT64-Gateway fungiert, gibt es guten Grund, einen Paketfilter zu installieren. Der Router ist nämlich zentrale Austauschpunkt für Daten unseres Lern- und Bastelnetzwerkes mit dem IPv6-Internet. Wir werden als schädlich erkannte Pakete grundsätzlich verwerfen und alle anderen überwiegend auf Plausibilität prüfen. Glücklicherweise nimmt uns das Connection Tracking den Großteil der Plausibilitätsprüfungen ab.

Zonen

Beim Schreiben komplexer Paketfilter verliert man leicht den Überblick. Eine Möglichkeit, dem Vorzubeugen, ist das Arbeiten mit Zonen. Eine Zone ist dann ein Bereich eines Netzwerkes, für den einheitliche Regeln gelten. Auch wir werden mit Zonen arbeiten, um eine einfache Abstraktion vom Konkreten zu erreichen. Dazu definieren wir eine Zone *Untrusted*, in der sich alle Systeme befinden, die nicht unter unserer Kontrolle stehen. Unter anderem der Teil des IPv6-Internets, der sich nicht in unserem Lern- und Bastelnetzwerk abspielt. Alles was am Link *internal* geschieht, halten wir für vertrauenswürdig, diese Zone trägt daher den Namen *Trusted*.

Zonen und Interfaces

Zonengrenzen befinden sich immer an Interfaces, und niemals inmitten eines Links. Zwei oder mehr Zonen können sich daher auch nur innerhalb eines Nodes treffen. In Abbildung 7.6 sind die Zonen zu sehen, in denen sich *fuzzball* befindet.

Der Weg ins IPv6-Internet führt über das Interface *sixxs*, es zeigt in die Zone Untrusted. Daher sind Pakete die über *sixxs* ins System gelangen besonders streng zu filtern. Ebenfalls der Zone Untrusted wird das Interface *nat64* zugeordnet. Pakete die über dieses Interface in das System gelangen, stammen aus dem IPv4-Internet, welches ebenfalls nicht unter unserer Kontrolle steht. Pakete die über *eth1* empfangen werden, kom-

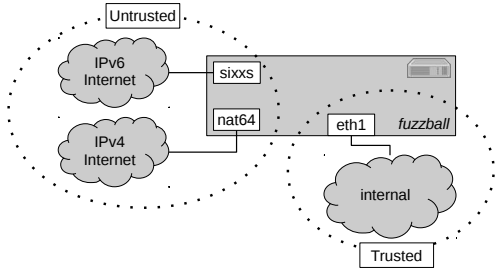


Abbildung 7.6
Zonen von *fuzzball*

men aus der Zone Trusted, denn sie stammen von Systemen die wir unter Kontrolle haben.

Zonen und Chains

Die Zonen lassen sich nicht ohne weiteres auf die eingebauten Chains übertragen. So laufen alle eingehenden Pakete durch die Chain INPUT, unabhängig davon ob sie aus einer Trusted- oder Untrusted-Zone stammen. In den Regeln muss daher das betroffene Interface berücksichtigt werden. Darum ist es manchmal sinnvoll, nutzerdefinierte Chains für eine Zone zu erstellen, und diese abhängig vom Interface einzuhängen. An anderen Stellen wiederum ist es praktischer, für bestimmte Protokolle nutzerdefinierte Chains anzulegen. Die können dann in allen betroffenen Zonen verwendet werden. Der Paketfilter von *fuzzball* wird von beiden Techniken Gebrauch machen.

Zonenübertritte

Oft müssen Pakete ihre Zone verlassen um ein Ziel in einer anderen Zone zu erreichen. Solche Pakete, die von einer Zone in eine andere befördert werden wollen, sollen ebenfalls vom Paketfilter geprüft werden. Sie passieren die Chain FORWARD. In dieser Chain treffen mehrere Zonen aufeinander. Um herauszufinden, von welcher Zone in welche andere Zone ein Paket befördert werden möchte, müssen wir auf die betroffenen Interfaces schauen. Ein- und ausgehendes Interface sind in den Regeln der FORWARD-Chain wichtige Parameter.

Vorhandene Regeln entfernen

Wir beginnen nun mit dem Aufbau des Paketfilters für *fuzzball*. Der erste Schritt ist das Entfernen aller möglicherweise bestehenden Regeln:

```
root@fuzzball:~# ip6tables --flush
root@fuzzball:~# ip6tables --flush --table mangle
root@fuzzball:~# ip6tables --delete-chain
root@fuzzball:~# ip6tables --delete-chain --table mangle
```

Vom Erfolg der Maßnahme werden wir uns überzeugen:

```
root@fuzzball:~# ip6tables --list
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

Wir sehen, dass in den Chains INPUT, FORWARD und OUTPUT keine Regeln existieren.

Default Policies

Wir können also nun dem Aufbau des Paketfilters beginnen. Zuerst legen wir die Default Policies der Chains INPUT und FORWARD fest. Grundsätzlich sollen Pakete verworfen werden. Wir betreiben in diesen Chains also Whitelisting:

```
root@fuzzball:~# ip6tables --policy INPUT DROP
root@fuzzball:~# ip6tables --policy FORWARD DROP
```

Auch hier lassen wir uns das Ergebnis der letzten Kommandos anzeigen:

```
root@fuzzball:~# ip6tables --list
Chain INPUT (policy DROP)
target     prot opt source               destination

Chain FORWARD (policy DROP)
target     prot opt source               destination
```

Die Default Policies verweisen jetzt auf das Target DROP für ein kommentarloses Verwerfen von Paketen.

Loopback Interface

Das wohl einzige Interface auf *fuzzball*, welches uneingeschränktes Vertrauen genießt, ist das Loopback Interface. Pakete von und zum Loopback Interface dürfen den Paketfilter ungehindert passieren:

```
root@fuzzball:~# ip6tables --append INPUT --in-interface ⤸
    lo --jump ACCEPT
root@fuzzball:~# ip6tables --append OUTPUT ⤸
    --out-interface lo --jump ACCEPT
```

Chain für verworfene Pakete

Für Pakete die verworfen werden sollen, aber dennoch von Interesse sein könnten, erstellen wir eine nutzerdefinierte Chain. Sie dient anderen Regeln als Target. Die Chain schreibt Pakete mit dem Level *Notice* ins Systemlog, anschließend verwirft sie die Pakete:

```
root@fuzzball:~# ip6tables --new-chain trashlog
root@fuzzball:~# ip6tables --append trashlog --jump LOG ⤸
    --log-level notice --log-prefix "TRASHLOG: "
root@fuzzball:~# ip6tables --append trashlog --jump DROP
```

Ein kurzer Blick auf die Chain zeigt, ob sie wie gewünscht angelegt wurde:

```
root@fuzzball:~# ip6tables --list trashlog
Chain trashlog (0 references)
target     prot opt source               destination
LOG        all      anywhere             anywhere ⤸
               LOG level notice prefix 'TRASHLOG: '
DROP       all      anywhere             anywhere
```

Connection Tracking nutzen

Pakete bestehender Verbindungen und ihre verwandten Pakete sollen den Paketfilter ungehindert passieren. Die Plausibilitätsprüfung für diese Pakete hat das Connection Tracking bereits erledigt. Wir fügen die entsprechende Regel der INPUT-Chain hinzu:

```
root@fuzzball:~# ip6tables --append INPUT --match ⤸
    conntrack --ctstate ESTABLISHED,RELATED --jump ACCEPT
```

Ungültige Pakete verwerfen

Eingehende ungültige Pakete, egal aus welcher Zone sie stammen, werden wir verwerfen. Da ungültige Pakete manchmal auf Fehlkonfigurationen hinweisen, lassen wir sie zur Zwecken der Fehlerbehebung ins Systemlog schreiben:

```
root@fuzzball:~# ip6tables --append INPUT --match ⤸
    conntrack --ctstate INVALID --jump trashlog
```

Ohne Angabe eines Interfaces, trifft diese Regel auf alle eingehenden Pakete, und damit auch auf alle Zonen, zu.

Extension Header

Der Routing Header Typ 0 wurde als gefährlich eingestuft und gilt als veraltet. Auch andere Extension Header könnten, abhängig von der Reihenfolge ihres Auftretens und ihrer Parameter, in Zukunft unerwünscht sein. Um den Überblick nicht zu verlieren, erstellen wir eine nutzerdefinierte Chain namens *bad-eh*, in der alle unerwünschten Extension Header benannt werden. Der Routing Header Typ 0 wird mit einer passenden Regel verworfen:

```
root@fuzzball:~# ip6tables --new-chain bad-eh
root@fuzzball:~# ip6tables --append bad-eh --match rt ⤸
    --rt-type 0 --jump DROP
```

Andere Regeln können bei Bedarf später hinzugefügt werden. Damit die Regel wirksam wird, hängen wir die eben erstellte Chain in die INPUT-Chain ein:

```
root@fuzzball:~# ip6tables --append INPUT --jump bad-eh
```

Neighbor Discovery

Sowohl auf den Links der Trusted-Zone, als auch auf den Links der Untrusted-Zone, benötigen wir das Neighbor Discovery Protocol für die Selbstorganisation des Links. Dazu schreiben wir uns eine nutzerdefinierte Chain *ndp-minimal* mit den nötigen Regeln für NDP:

```
root@fuzzball:~# ip6tables --new-chain ndp-minimal
root@fuzzball:~# ip6tables --append ndp-minimal ⤸
    --protocol icmpv6 --icmpv6-type ⤸
    neighbor-solicitation --match hl --hl-eq 255 --jump ⤸
    ACCEPT
root@fuzzball:~# ip6tables --append ndp-minimal ⤸
    --protocol icmpv6 --icmpv6-type ⤸
    neighbor-advertisement --match hl --hl-eq 255 ⤸
    --jump ACCEPT
```

Auch auf das Umleitungen von Pakete lassen wir uns ein:

```
root@fuzzball:~# ip6tables --append ndp-minimal ⤸
    --protocol icmpv6 --icmpv6-type redirect --match hl ⤸
    --hl-eq 255 --jump ACCEPT
```

MLDv1 und MLDv2 sind vorgeschrieben und notwendig, darum erlauben wir die beiden Protokolle ebenfalls:

```
# Typ 130: Multicast Listener Query (MLDv1, MLDv2)
root@fuzzball:~# ip6tables --append ndp-minimal ⤸
    --protocol icmpv6 --icmpv6-type 130 --match hl ⤸
    --hl-eq 1 --jump ACCEPT
# Typ 131: Multicast Listener Report (MLDv1)
root@fuzzball:~# ip6tables --append ndp-minimal ⤸
    --protocol icmpv6 --icmpv6-type 131 --match hl ⤸
    --hl-eq 1 --jump ACCEPT
# Typ 132: Multicast Listener Done (MLDv1)
root@fuzzball:~# ip6tables --append ndp-minimal ⤸
    --protocol icmpv6 --icmpv6-type 132 --match hl ⤸
    --hl-eq 1 --jump ACCEPT
# Typ 143: Multicast Listener Report (MLDv2)
root@fuzzball:~# ip6tables --append ndp-minimal ⤸
    --protocol icmpv6 --icmpv6-type 143 --match hl ⤸
    --hl-eq 1 --jump ACCEPT
```

Danach hängen wir die Chain ndp-minimal in die INPUT-Chain ein, jedoch nur für eintreffende ICMPv6-Pakete:

```
root@fuzzball:~# ip6tables --append INPUT --protocol ⤸
    icmpv6 --jump ndp-minimal
```

Auch von dieser Regel sind wieder alle Zonen betroffen.

Router Solicitations

Als nächstes werden wir eingehende Router Solicitations von der Trusted-Zone erlauben, denn für diese ist *fuzzball* der zuständige Router:

```
root@fuzzball:~# ip6tables --append INPUT --in-interface ⤸
    eth1 --protocol icmpv6 --icmpv6-type ⤸
    router-solicitation --match hl --hl-eq 255 --jump ⤸
    ACCEPT
```

Resolving DNS Server

In den Router Advertisements verteilen wir die Adresse des Resolving DNS Servers. Dessen Dienste sollten wir für die Trusted-Zone ebenfalls freigeben:

```
root@fuzzball:~# ip6tables --append INPUT --in-interface ⤸
    eth1 --protocol udp --destination-port 53 --match ⤸
    conntrack --ctstate NEW --jump ACCEPT
root@fuzzball:~# ip6tables --append INPUT --in-interface ⤸
    eth1 --protocol tcp --destination-port 53 --match ⤸
    conntrack --ctstate NEW --jump ACCEPT
```

Forwarding filtern

Widmen wir uns nun der Chain FORWARD. Durch sie laufen alle Pakete hindurch, die von einem Interface zu einem anderen müssen. In unserem Fall bedeutet das auch immer einen

Zonenwechsel von Trusted zu Untrusted oder umgekehrt. Wäre *fuzzball* in weiteren Zonen vertreten, dann würde durch diese Chain auch der Verkehr zwischen den anderen Zonen laufen.

Connection Tracking

Auch für weitergeleitete Pakete kann Netfilter ein Connection Tracking durchführen. Das nimmt uns eine Menge Arbeit beim definieren von Regeln ab. Gleichzeitig erhöht ein frühzeitiges Durchlassen von unschädlichen Paketen die Effizienz der Regelabarbeitung. Bestehende Verbindungen und allen damit verwandten Paketen erlauben wir deshalb auch in der FORWARD-Chain das Passieren:

```
root@fuzzball:~# ip6tables --append FORWARD --match ⤸
     conntrack --ctstate ESTABLISHED,RELATED --jump ACCEPT
```

Extension Header

Gefährliche Extension Header werden wir auch beim Forwarding nicht akzeptieren. Darum hängen wir die nutzerdefinierte Chain bad-eh auch in der FORWARD-Chain ein:

```
root@fuzzball:~# ip6tables --append FORWARD --jump bad-eh
```

Nutzerdefinierte Chain für ICMPv6

Bei den Upper Layer Protocols hat sich im Bereich Forwarding nichts Wesentliches im Vergleich zu IPv4 geändert. Den Schwerpunkt des Filterns legen wir deshalb auf ICMPv6 fest. Wir starten mit einer neuen nutzerdefinierten Chain namens *icmpv6-filter*:

```
root@fuzzball:~# ip6tables --new-chain icmpv6-filter
```

Link-local Addresses

Link-local Addresses besitzen bekanntlich nur auf ihrem eigenen Link Gültigkeit. Deshalb dürfen sie niemals weitergeleitet werden. Pakete die als Quell- oder Zieladresse eine Link-local Addresses enthalten, werden daher verworfen:

```
root@fuzzball:~# ip6tables --append icmpv6-filter ⤸
     --source fe80::/10 --jump DROP
root@fuzzball:~# ip6tables --append icmpv6-filter ⤸
     --destination fe80::/10 --jump DROP
```

Echo Requests

Echo Requests sind ein nützliches Werkzeug bei der Fehlersuche, darum werden wir sie nicht grundsätzlich verbieten. Einige Einschränkungen werden wir aber machen. Nodes aus der Trusted-Zone sollen Echo Request an jeden versenden dürfen:

```
root@fuzzball:~# ip6tables --append icmpv6-filter ⤸
     --source 2a01:198:200:8a23::/64 --protocol icmpv6 ⤸
     --icmpv6-type echo-request --match conntrack ⤸
     --ctstate NEW --jump ACCEPT
```

Aber nur bestimmte Nodes aus der Trusted-Zone dürfen vom Rest der Welt mit Echo Requests angesprochen werden. Die Liste können Sie beliebig erweitern. Im Workshop geben wir *lynx* frei, *felis* bleibt für Echo Requests unerreichbar.

```
root@fuzzball:~# ip6tables --append icmpv6-filter ⤸
     --destination 2a01:198:200:8a23:200:ff:fe60:d1e ⤸
     --protocol icmpv6 --icmpv6-type echo-request ⤸
     --match conntrack --ctstate NEW --jump ACCEPT
```

Das Präfix und die Adresse in den Parametern von `ip6tables` müssen Sie natürlich an Ihre eigenen Gegebenheiten anpassen.

Echo Replies

Die Antwortpakete (Echo Replies) für die eben erstellten Regeln werden dank Connection Tracking frühzeitig erlaubt. Andere Echo Replies sind davon allerdings nicht betroffen. Zum Beispiel Echo Replies die an eine Multicast Address verschickt werden. So ein Verhalten ist schädlich und kann von Betroffenen sogar als Angriff aufgefasst werden. Echo Replies auf Multicast Addresses werden wir daher grundsätzlich verwerfen:

```
root@fuzzball:~# ip6tables --append icmpv6-filter ⤸
     --destination ff00::/8 --protocol icmpv6 ⤸
     --icmpv6-type echo-reply --jump DROP
```

Error Messages

ICMPv6 Error Messages für bestehende Verbindungen werden vom Connection Tracking erkannt und wurden von uns bereits erlaubt. Darüber hinaus sollen die Nodes in der Trusted-Zone die Möglichkeit erhalten, auch ungefragt Fehlermeldungen zu verschicken.

Wir beginnen mit der wichtigsten Fehlermeldung wenn es um Path MTU Discovery geht, nämlich der Packet Too Big Error Message:

```
root@fuzzball:~# ip6tables --append icmpv6-filter ⤸
    --source 2a01:198:200:8a23::/64 --protocol icmpv6 ⤸
    --icmpv6-type packet-too-big --jump ACCEPT
```

Auch das Überschreiten der Zeitgrenze für den Zusammenbau von Fragmenten dürfen Nodes aus der Trusted-Zone anzeigen:

```
root@fuzzball:~# ip6tables --append icmpv6-filter ⤸
    --source 2a01:198:200:8a23::/64 --protocol icmpv6 ⤸
    --icmpv6-type ttl-zero-during-reassembly --jump ACCEPT
```

Probleme beim Auswerten von Headern oder Parametern sollen ebenso den Paketfilter passieren dürfen:

```
root@fuzzball:~# ip6tables --append icmpv6-filter ⤸
    --source 2a01:198:200:8a23::/64 --protocol icmpv6 ⤸
    --icmpv6-type bad-header --jump ACCEPT
root@fuzzball:~# ip6tables --append icmpv6-filter ⤸
    --source 2a01:198:200:8a23::/64 --protocol icmpv6 ⤸
    --icmpv6-type unknown-header-type --jump ACCEPT
root@fuzzball:~# ip6tables --append icmpv6-filter ⤸
    --source 2a01:198:200:8a23::/64 --protocol icmpv6 ⤸
    --icmpv6-type unknown-option --jump ACCEPT
```

In der Trusted-Zone befinden sich nicht nur Hosts, sondern auch *fuzzball* als Router ist mit einem Interface in der Zone vertreten. Daher erlauben wir auch die Fehlermeldungen, die für Router typisch sind, nämlich Destination Unreachable und Hop Limit Exceeded:

```
root@fuzzball:~# ip6tables --append icmpv6-filter ⤸
    --source 2a01:198:200:8a23::/64 --protocol icmpv6 ⤸
    --icmpv6-type destination-unreachable --jump ACCEPT
root@fuzzball:~# ip6tables --append icmpv6-filter ⤸
    --source 2a01:198:200:8a23::/64 --protocol icmpv6 ⤸
    --icmpv6-type ttl-zero-during-transit --jump ACCEPT
```

Neighbor Discovery Protocol

Auf keinen Fall dürfen ICMPv6-Nachrichten den Router passieren, die außerhalb des lokalen Links nichts zu suchen haben. Als erstes verwerfen wir NDP:

```
root@fuzzball:~# ip6tables --append icmpv6-filter ⤸
     --protocol icmpv6 --icmpv6-type ⤸
     neighbor-solicitation --jump DROP
root@fuzzball:~# ip6tables --append icmpv6-filter ⤸
     --protocol icmpv6 --icmpv6-type ⤸
     neighbor-advertisement --jump DROP
root@fuzzball:~# ip6tables --append icmpv6-filter ⤸
     --protocol icmpv6 --icmpv6-type router-solicitation ⤸
     --jump DROP
root@fuzzball:~# ip6tables --append icmpv6-filter ⤸
     --protocol icmpv6 --icmpv6-type ⤸
     router-advertisement --jump DROP
```

Auch Umleitungen haben beim Forwarding nichts verloren:

```
root@fuzzball:~# ip6tables --append icmpv6-filter ⤸
     --protocol icmpv6 --icmpv6-type redirect --jump DROP
```

Die Protokolle MLDv1 und MLDv2 sind zwischen den Zonen ebenfalls unerwünscht.

```
# Typ 130: Multicast Listener Query (MLDv1, MLDv2)
root@fuzzball:~# ip6tables --append icmpv6-filter ⤸
     --protocol icmpv6 --icmpv6-type 130 --jump DROP
# Typ 131: Multicast Listener Report (MLDv1)
root@fuzzball:~# ip6tables --append icmpv6-filter ⤸
     --protocol icmpv6 --icmpv6-type 131 --jump DROP
# Typ 132: Multicast Listener Done (MLDv1)
root@fuzzball:~# ip6tables --append icmpv6-filter ⤸
     --protocol icmpv6 --icmpv6-type 132 --jump DROP
# Typ 143: Multicast Listener Report (MLDv2)
root@fuzzball:~# ip6tables --append icmpv6-filter ⤸
     --protocol icmpv6 --icmpv6-type 143 --jump DROP
```

Router Renumbering

ICMPv6-Nachrichten zur Neunummerierung von Netzwerken sollen auch nicht weitergeleitet werden:

```
# Typ 138:
root@fuzzball:~# ip6tables --append icmpv6-filter ⤸
     --protocol icmpv6 --icmpv6-type 147 --jump DROP
```

Node-Informationen

Mit den ICMPv6-Nachrichten *Node Information Query* kann man von einem Node Informationen einholen, zum Beispiel seinen Hostnamen. Der Node antwortet dann mit einem *Node Information Reply*, der die gewünschten Informationen enthält. Das Verfahren ist in RFC 4620 [CH06] spezifiziert und gilt als experimentell. Solange wir dafür keine Verwendung haben, verwerfen wir die zugehörigen Pakete deshalb:

```
# Typ 139: Node Information Query
root@fuzzball:~# ip6tables --append icmpv6-filter ⤸
    --protocol icmpv6 --icmpv6-type 139 --jump DROP
# Typ 140: Node Information Reply
root@fuzzball:~# ip6tables --append icmpv6-filter ⤸
    --protocol icmpv6 --icmpv6-type 140 --jump DROP
```

Mobile IPv6

Da wir kein Mobile IPv6 einsetzen, werden wir auch alle entsprechenden IPCMv6-Nachrichten verwerfen:

```
# Typ 144: Home Agent Address Discovery Request
root@fuzzball:~# ip6tables --append icmpv6-filter ⤸
    --protocol icmpv6 --icmpv6-type 144 --jump DROP
# Typ 145: Home Agent Address Discovery Reply
root@fuzzball:~# ip6tables --append icmpv6-filter ⤸
    --protocol icmpv6 --icmpv6-type 145 --jump DROP
# Typ 146 Mobile Prefix Solicitation
root@fuzzball:~# ip6tables --append icmpv6-filter ⤸
    --protocol icmpv6 --icmpv6-type 146 --jump DROP
# Typ 147: Mobile Prefix Advertisement
root@fuzzball:~# ip6tables --append icmpv6-filter ⤸
    --protocol icmpv6 --icmpv6-type 147 --jump DROP
```

Alle weiteren ICMPv6-Pakete verwerfen wir:

```
root@fuzzball:~# ip6tables --append icmpv6-filter --jump ⤸
    DROP
```

Sollte es Probleme beim Forwarding geben, und der Verdacht fällt auf verworfene ICMPv6-Pakete, dann ersetzen wir das Target der zuletzt eingefügten Regeln durch trashlog. Dann werden die Pakete weiterhin verworfen, aber wir können den Vorgang im Systemlog beobachten. Mit den Erkenntnissen lassen sich dann gegebenenfalls neue Regeln formulieren.

Zum Abschluss hängen wir die nutzerdefinierte Chain icmpv6-filter noch in die FORWARD-Chain ein:

```
root@fuzzball:~# ip6tables --append FORWARD --protocol ⤸
    icmpv6 --jump icmpv6-filter
```

Upper Layer Protocols

Das Filtern von Upper Layer Protocols ist ein Thema, welches weder zum Titel des Workshops passt, noch in dessen Rahmen ausreichend behandelt werden könnte. Die folgenden Regeln erlauben einfach jede neue Verbindung aus der Trusted-Zone in die Untrusted-Zone:

Datei: /etc/rc.local

```
1 #!/bin/sh -e
2 #
3 # By default this script does nothing.
4
5 ip6tables-restore < /etc/packetfilter
6
7 exit 0
```

Abbildung 7.7 Skript rc.local

```
root@fuzzball:~# ip6tables --append FORWARD ⤸
     --in-interface eth1 --out-interface sixxs --match ⤸
     conntrack --ctstate NEW --jump ACCEPT
root@fuzzball:~# ip6tables --append FORWARD ⤸
     --in-interface eth1 --out-interface nat64 --match ⤸
     conntrack --ctstate NEW --jump ACCEPT
```

Betrachten Sie diese beiden Regeln bitte als Notbehelf im Rahmen des Workshops, und nicht als Ersatz für einen durchdachten Paketfilter. Dort wo dieses Kapitel endet, fängt das klassische Paketfiltern der höheren Protokolle an.

Regeln sichern

Wie schon bei *lynx* durchgeführt, werden wir auch für *fuzzball* die Regeln sichern. Nicht nur des effizienteren Formates wegen, sondern auch um alle erstellten Regeln im Überblick zu behalten. Wir leiten die Ausgabe von `ip6tables-save` wieder in eine Datei um:

```
root@fuzzball:~# ip6tables-save > /etc/packetfilter
```

Die Datei `/etc/packetfilter` von *fuzzball* ist in Anhang D *Paketfilter von fuzzball* zu sehen.

Regeln wiederherstellen

Bei jedem Systemstart werden wir den Paketfilter in den Kernel laden, damit *fuzzball* sofort und sicher seine Arbeit aufnehmen kann. Dazu verändern wir die Datei `/etc/rc.local` wie in Abbildung 7.7 gezeigt.

Verworfene Pakete mitschreiben

Viele der vorgestellten Regeln verwerfen ungewollte Pakete kommentarlos. Im produktiven Betrieb ist es häufig notwendig, einen Kompromiss zwischen dem Mitschreiben von Paketen und der Geschwindigkeit des Paketfilters zu finden. In unserem Lern- und Bastelnetzwerk sind wir diesen Zwängen nicht ausgesetzt, und könnten theoretisch jedes verworfene

Paket dokumentieren. Dazu müssen nur die Targets der entsprechenden Regeln auf die nutzerdefinierte Chain trashlog zeigen.

Systemlog auswerten

Im Folgenden Beispiel wurde die Regel für eingehende Echo Requests auf trashlog umgeleitet. Ein Echo Request von außerhalb wurde daraufhin verworfen, aber vorher im Systemlog hinterlegt. Einen Blick auf das Systemlog erhalten wir mit dem Kommando `dmesg`, welches hier ausgeführt wurde:

```
root@fuzzball:~# dmesg
✂ TRASHLOG: IN=sixxs OUT=eth1 MAC= ⤶
      SRC=2001:07f8:0033:0000:0000:a105:7821:0001 ⤶
      DST=2a01:0198:0200:8a23:0000:0000:0bad:affe LEN=104 ⤶
      TC=0 HOPLIMIT=59 FLOWLBL=0 PROTO=ICMPv6 TYPE=128 ⤶
      CODE=0 ID=7609 SEQ=1
```

Die gezeigte Zeile wurde vom verworfenen Paket erzeugt. Alle wichtigen Informationen sind enthalten, so dass Vorfälle nachvollzogen werden können. In diesem Fall wurde ein Echo Request mit dem Identifier 7609 und der Sequenznummer 1 verworfen.

Erfahrungen sammeln

Beschäftigen Sie sich ruhig noch eine Weile mit dem Paketfilter von *fuzzball*. Schreiben Sie Pakete mit, und versuchen Sie anhand des Systemlogs nachzuvollziehen, was passiert ist.

Anhang

A | Spickzettel Zahlensysteme

Binär	Dezimal	Hexadezimal
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	a
1011	11	b
1100	12	c
1101	13	d
1110	14	e
1111	15	f

```
>>> mysrcip="2a01:198:200:8a23::1"
>>> mydstip="2a01:198:200:8a23:200:ff:fe60:d1e"
>>> mypayload="AAAAAAAA"
>>> myheader=IPv6(src=mysrcip,dst=mydstip,plen=16)
>>> myicmpv6=ICMPv6EchoRequest(cksum=0x7d2b)
>>> myfragment1=IPv6ExtHdrFragment(offset=0,m=1,id=1337,nh=58)
>>> myfragment2=IPv6ExtHdrFragment(offset=8,m=0,id=1337,nh=58)
>>> send(myheader/myfragment1/myicmpv6)
>>> send(myheader/myfragment2/mypayload)
```

C | Paketfilter von lynx

```
1  *mangle
2  :PREROUTING ACCEPT [0:0]
3  :INPUT ACCEPT [0:0]
4  :FORWARD ACCEPT [0:0]
5  :OUTPUT ACCEPT [0:0]
6  :POSTROUTING ACCEPT [0:0]
7  COMMIT
8  #
9  *filter
10 :INPUT DROP [0:0]
11 :FORWARD DROP [0:0]
12 :OUTPUT ACCEPT [0:0]
13 :ndp-slaac - [0:0]
14 :trashlog - [0:0]
15 -A INPUT -i lo -j ACCEPT
16 -A INPUT -m conntrack --ctstate INVALID -j trashlog
17 -A INPUT -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
18 -A INPUT -p ipv6-icmp -j ndp-slaac
19 -A INPUT -s fe80::/10 -d fe80::/10 -p ipv6-icmp -m icmp6 --icmpv6-type ⤸
       128 -m conntrack --ctstate NEW -j ACCEPT
20 -A INPUT -s fe80::/10 -p tcp -m tcp --dport 22 -m conntrack --ctstate ⤸
       NEW -j ACCEPT
21 -A OUTPUT -o lo -j ACCEPT
22 -A ndp-slaac -p ipv6-icmp -m icmp6 --icmpv6-type 133 -m hl --hl-eq 255 ⤸
       -j ACCEPT
23 -A ndp-slaac -p ipv6-icmp -m icmp6 --icmpv6-type 134 -m hl --hl-eq 255 ⤸
       -j ACCEPT
24 -A ndp-slaac -p ipv6-icmp -m icmp6 --icmpv6-type 135 -m hl --hl-eq 255 ⤸
       -j ACCEPT
25 -A ndp-slaac -p ipv6-icmp -m icmp6 --icmpv6-type 136 -m hl --hl-eq 255 ⤸
       -j ACCEPT
26 -A ndp-slaac -p ipv6-icmp -m icmp6 --icmpv6-type 137 -m hl --hl-eq 255 ⤸
       -j ACCEPT
27 -A ndp-slaac -p ipv6-icmp -m icmp6 --icmpv6-type 130 -m hl --hl-eq 1 -j ⤸
       ACCEPT
28 -A ndp-slaac -p ipv6-icmp -m icmp6 --icmpv6-type 131 -m hl --hl-eq 1 -j ⤸
       ACCEPT
29 -A ndp-slaac -p ipv6-icmp -m icmp6 --icmpv6-type 132 -m hl --hl-eq 1 -j ⤸
       ACCEPT
30 -A ndp-slaac -p ipv6-icmp -m icmp6 --icmpv6-type 143 -m hl --hl-eq 1 -j ⤸
       ACCEPT
31 -A trashlog -j LOG --log-prefix "TRASHLOG: " --log-level 5
32 -A trashlog -j DROP
33 COMMIT
```

```
1  *mangle
2  :PREROUTING ACCEPT [0:0]
3  :INPUT ACCEPT [0:0]
4  :FORWARD ACCEPT [0:0]
5  :OUTPUT ACCEPT [0:0]
6  :POSTROUTING ACCEPT [0:0]
7  COMMIT
8  #
9  *filter
10 :INPUT DROP [0:0]
11 :FORWARD DROP [0:0]
12 :OUTPUT ACCEPT [0:0]
13 :bad-eh - [0:0]
14 :icmpv6-filter - [0:0]
15 :ndp-minimal - [0:0]
16 :trashlog - [0:0]
17 -A INPUT -i lo -j ACCEPT
18 -A INPUT -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
19 -A INPUT -m conntrack --ctstate INVALID -j trashlog
20 -A INPUT -p ipv6-icmp -j ndp-minimal
21 -A INPUT -i eth1 -p ipv6-icmp -m icmp6 --icmpv6-type 133 -m hl --hl-eq ⤸
       255 -j ACCEPT
22 -A INPUT -i eth1 -p udp -m udp --dport 53 -m conntrack --ctstate NEW -j ⤸
       ACCEPT
23 -A INPUT -i eth1 -p tcp -m tcp --dport 53 -m conntrack --ctstate NEW -j ⤸
       ACCEPT
24 -A FORWARD -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
25 -A FORWARD -p ipv6-icmp -j icmpv6-filter
26 -A FORWARD -i eth1 -o sixxs -m conntrack --ctstate NEW -j ACCEPT
27 -A FORWARD -i eth1 -o nat64 -m conntrack --ctstate NEW -j ACCEPT
28 -A OUTPUT -o lo -j ACCEPT
29 -A bad-eh -m rt --rt-type 0 --rt-segsleft 0 -j DROP
30 -A icmpv6-filter -s fe80::/10 -j DROP
31 -A icmpv6-filter -d fe80::/10 -j DROP
32 -A icmpv6-filter -s 2a01:198:200:8a23::/64 -p ipv6-icmp -m icmp6 ⤸
       --icmpv6-type 128 -m conntrack --ctstate NEW -j ACCEPT
33 -A icmpv6-filter -d 2a01:198:200:8a23:200:ff:fe60:d1e/128 -p ipv6-icmp ⤸
       -m icmp6 --icmpv6-type 128 -m conntrack --ctstate NEW -j ACCEPT
34 -A icmpv6-filter -d ff00::/8 -p ipv6-icmp -m icmp6 --icmpv6-type 129 -j ⤸
       DROP
35 -A icmpv6-filter -s 2a01:198:200:8a23::/64 -p ipv6-icmp -m icmp6 ⤸
       --icmpv6-type 2 -j ACCEPT
36 -A icmpv6-filter -s 2a01:198:200:8a23::/64 -p ipv6-icmp -m icmp6 ⤸
       --icmpv6-type 3/1 -j ACCEPT
37 -A icmpv6-filter -s 2a01:198:200:8a23::/64 -p ipv6-icmp -m icmp6 ⤸
       --icmpv6-type 4/0 -j ACCEPT
38 -A icmpv6-filter -s 2a01:198:200:8a23::/64 -p ipv6-icmp -m icmp6 ⤸
       --icmpv6-type 4/1 -j ACCEPT
39 -A icmpv6-filter -s 2a01:198:200:8a23::/64 -p ipv6-icmp -m icmp6 ⤸
       --icmpv6-type 4/2 -j ACCEPT
40 -A icmpv6-filter -s 2a01:198:200:8a23::/64 -p ipv6-icmp -m icmp6 ⤸
       --icmpv6-type 1 -j ACCEPT
```

```
-A icmpv6-filter -s 2a01:198:200:8a23::/64 -p ipv6-icmp -m icmp6 ⤶
    --icmpv6-type 3/0 -j ACCEPT
-A icmpv6-filter -p ipv6-icmp -m icmp6 --icmpv6-type 135 -j DROP
-A icmpv6-filter -p ipv6-icmp -m icmp6 --icmpv6-type 136 -j DROP
-A icmpv6-filter -p ipv6-icmp -m icmp6 --icmpv6-type 133 -j DROP
-A icmpv6-filter -p ipv6-icmp -m icmp6 --icmpv6-type 134 -j DROP
-A icmpv6-filter -p ipv6-icmp -m icmp6 --icmpv6-type 137 -j DROP
-A icmpv6-filter -p ipv6-icmp -m icmp6 --icmpv6-type 130 -j DROP
-A icmpv6-filter -p ipv6-icmp -m icmp6 --icmpv6-type 131 -j DROP
-A icmpv6-filter -p ipv6-icmp -m icmp6 --icmpv6-type 132 -j DROP
-A icmpv6-filter -p ipv6-icmp -m icmp6 --icmpv6-type 143 -j DROP
-A icmpv6-filter -p ipv6-icmp -m icmp6 --icmpv6-type 147 -j DROP
-A icmpv6-filter -p ipv6-icmp -m icmp6 --icmpv6-type 139 -j DROP
-A icmpv6-filter -p ipv6-icmp -m icmp6 --icmpv6-type 140 -j DROP
-A icmpv6-filter -p ipv6-icmp -m icmp6 --icmpv6-type 144 -j DROP
-A icmpv6-filter -p ipv6-icmp -m icmp6 --icmpv6-type 145 -j DROP
-A icmpv6-filter -p ipv6-icmp -m icmp6 --icmpv6-type 146 -j DROP
-A icmpv6-filter -p ipv6-icmp -m icmp6 --icmpv6-type 147 -j DROP
-A icmpv6-filter -j DROP
-A ndp-minimal -p ipv6-icmp -m icmp6 --icmpv6-type 135 -m hl --hl-eq 255 ⤶
    -j ACCEPT
-A ndp-minimal -p ipv6-icmp -m icmp6 --icmpv6-type 136 -m hl --hl-eq 255 ⤶
    -j ACCEPT
-A ndp-minimal -p ipv6-icmp -m icmp6 --icmpv6-type 137 -m hl --hl-eq 255 ⤶
    -j ACCEPT
-A ndp-minimal -p ipv6-icmp -m icmp6 --icmpv6-type 130 -m hl --hl-eq 1 ⤶
    -j ACCEPT
-A ndp-minimal -p ipv6-icmp -m icmp6 --icmpv6-type 131 -m hl --hl-eq 1 ⤶
    -j ACCEPT
-A ndp-minimal -p ipv6-icmp -m icmp6 --icmpv6-type 132 -m hl --hl-eq 1 ⤶
    -j ACCEPT
-A ndp-minimal -p ipv6-icmp -m icmp6 --icmpv6-type 143 -m hl --hl-eq 1 ⤶
    -j ACCEPT
-A trashlog -j LOG --log-prefix "TRASHLOG: " --log-level 5
-A trashlog -j DROP
COMMIT
```

Abkürzungsverzeichnis

6rd	IPv6 rapid deployment
ARP	Address Resolution Protocol
ARPA	Advanced Research Projects Agency
AYIYA	Anything In Anything
BOOTP	Bootstrap Protocol
CERN	European Organization for Nuclear Research
CGN	Carrier Grade NAT
CIDR	Classless Inter-Domain Routing
CPE	Customer Premise Equipment
CRC	Cyclic Redundancy Check
DARPA	Defense Advanced Research Projects Agency
DFN-Verein	Verein zur Förderung eines Deutschen Forschungsnetzes
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
DNSSEC	Domain Name System Security Extensions
DoS	Denial of Service
EUI-48	48-Bit Extended Unique Identifier
EUI-64	64-Bit Extended Unique Identifier
FTP	File Transfer Protocol
GNU	GNU's Not Unix
IAB	Internet Activities Board
IANA	Internet Assigned Numbers Authority
ICANN	Internet Corporation for Assigned Names and Numbers
ICCB	Internet Configuration Control Board

ICMPv6 ..... Internet Control Message Protocol Version 6
IETF ..... Internet Engineering Task Force
IP ..... Internet Protocol
IPsec ..... Internet Protocol Security
ISATAP ..... Intra-Site Automatic Tunnel Addressing Protocol
ISK ..... IP SixXS Kredit
ISO ..... International Organization for Standardization
ISP ..... Internet Service Provider
MAC ..... Media Access Control
mDNS ..... Multicast DNS
MIT ..... Massachusetts Institute of Technology
MLDv2 ..... Multicast Listener Discovery v2
MPLS ..... Multi Protocol Label Switching
MTU ..... Maximum Transmission Unit
NCP ..... Network Control Program
NDP ..... Neighbor Discovery Protocol
NIC ..... Network Interface Conroller
NTP ..... Network Time Protocol
OSI ..... Open Systems Interconnection
PAT ..... Port Address Translation
PoP ..... Point of Presence
PPP ..... Point-to-Point Protocol
QoS ..... Quality of Service
RDNSS ..... Resolving DNS Server
RFC ..... Request for Comments
RIPE NCC ..... Réseaux IP Européens Network Coordination Centre
RIR ..... Regional Internet Registry
SIIT ..... Stateless IP/ICMP Translation
SLAAC ..... Stateless Address Autoconfiguration
SNAT ..... Source Network Address Translation
SPI ..... Stateful Packet Inspection
SRI ..... Stanford Research Institute
SSH ..... Secure Shell

ST	Internet Stream Protocol
TCP	Transmission Control Protocol
The U	University of Utah
UCLA	University of California in Los Angeles
UCSB	University of California in Santa Barbara
UDP	User Datagram Protocol
VoIP	Voice over IP
VPN	Virtual Private Network
WinPcap	Windows Packet Capture
WLAN	Wireless LAN
WWW	World Wide Web

Stichwortverzeichnis

Literaturverzeichnis

[ACJR11] S. Amante, B. Carpenter, S. Jiang, and J. Rajahalme. IPv6 Flow Label Specification. RFC 6437 (Proposed Standard), November 2011.

[ASNN07] J. Abley, P. Savola, and G. Neville-Neil. Deprecation of Type 0 Routing Headers in IPv6. RFC 5095 (Proposed Standard), December 2007.

[BDH99] D. Borman, S. Deering, and R. Hinden. IPv6 Jumbograms. RFC 2675 (Proposed Standard), August 1999.

[BHB+10] C. Bao, C. Huitema, M. Bagnulo, M. Boucadair, and X. Li. IPv6 Addressing of IPv4/IPv6 Translators. RFC 6052 (Proposed Standard), October 2010.

[BMvB11] M. Bagnulo, P. Matthews, and I. van Beijnum. Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers. RFC 6146 (Proposed Standard), April 2011.

[Car05] B. Carpenter. RFC 1888 Is Obsolete. RFC 4048 (Informational), April 2005. Updated by RFC 4548.

[CCS96] I. Castineyra, N. Chiappa, and M. Steenstrup. The Nimrod Routing Architecture. RFC 1992 (Informational), August 1996.

[CD98] A. Conta and S. Deering. Generic Packet Tunneling in IPv6 Specification. RFC 2473 (Proposed Standard), December 1998.

[CDG06] A. Conta, S. Deering, and M. Gupta. Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification. RFC 4443 (Draft Standard), March 2006. Updated by RFC 4884.

[Cer73] V. Cerf. PARRY encounters the DOCTOR. RFC 439, January 1973.

[CH06] M. Crawford and B. Haberman. IPv6 Node Information Queries. RFC 4620 (Experimental), August 2006.

[CM01] B. Carpenter and K. Moore. Connection of IPv6 Domains via IPv4 Clouds. RFC 3056 (Proposed Standard), February 2001.

[Cra98] M. Crawford. Transmission of IPv6 Packets over Ethernet Networks. RFC 2464 (Proposed Standard), December 1998. Updated by RFC 6085.

[Cra00] M. Crawford. Router Renumbering for IPv6. RFC 2894 (Proposed Standard), August 2000.

[DFGL01] A. Durand, P. Fasano, I. Guardini, and D. Lento. IPv6 Tunnel Broker. RFC 3053 (Informational), January 2001.

[DFH99] S. Deering, W. Fenner, and B. Haberman. Multicast Listener Discovery (MLD) for IPv6. RFC 2710 (Proposed Standard), October 1999. Updated by RFCs 3590, 3810.

[DH98] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460 (Draft Standard), December 1998. Updated by RFCs 5095, 5722, 5871, 6437, 6564.

[DHJ+05] S. Deering, B. Haberman, T. Jinmei, E. Nordmark, and B. Zill. IPv6 Scoped Address Architecture. RFC 4007 (Proposed Standard), March 2005.

[Fau11] F. Le Faucheur. IP Router Alert Considerations and Usage. RFC 6398 (Best Current Practice), October 2011.

[GDLH12] W. George, C. Donley, C. Liljenstolpe, and L. Howard. IPv6 Support Required for All IP-Capable Nodes. RFC 6540 (Best Current Practice), April 2012.

[GTTD11] S. Gundavelli, M. Townsley, O. Troan, and W. Dec. Address Mapping of IPv6 Multicast Packets on Ethernet. RFC 6085 (Proposed Standard), January 2011.

[HC04] C. Huitema and B. Carpenter. Deprecating Site Local Addresses. RFC 3879 (Proposed Standard), September 2004.

[HCH06] H. Holbrook, B. Cain, and B. Haberman. Using Internet Group Management Protocol Version 3 (IGMPv3) and Multicast Listener Discovery Protocol Version 2 (MLDv2) for Source-Specific Multicast. RFC 4604 (Proposed Standard), August 2006.

[HD06] R. Hinden and S. Deering. IP Version 6 Addressing Architecture. RFC 4291 (Draft Standard), February 2006. Updated by RFCs 5952, 6052.

[HF12] N. Hilliard and D. Freedman. A Discard Prefix for IPv6. RFC 6666 (Informational), August 2012.

[HH05] R. Hinden and B. Haberman. Unique Local IPv6 Unicast Addresses. RFC 4193 (Proposed Standard), October 2005.

[HLS04] G. Huston, A. Lord, and P. Smith. IPv6 Address Prefix Reserved for Documentation. RFC 3849 (Informational), July 2004.

[HT02] B. Haberman and D. Thaler. Unicast-Prefix-based IPv6 Multicast Addresses. RFC 3306 (Proposed Standard), August 2002. Updated by RFCs 3956, 4489.

[Hui06] C. Huitema. Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs). RFC 4380 (Proposed Standard), February 2006. Updated by RFCs 5991, 6081.

[JPBM10] J. Jeong, S. Park, L. Beloeil, and S. Madanapalli. IPv6 Router Advertisement Options for DNS Configuration. RFC 6106 (Proposed Standard), November 2010.

[KK10] S. Kawamura and M. Kawashima. A Recommendation for IPv6 Address Text Representation. RFC 5952 (Proposed Standard), August 2010.

[LBB11] X. Li, C. Bao, and F. Baker. IP/ICMP Translation Algorithm. RFC 6145 (Proposed Standard), April 2011. Updated by RFC 6791.

[Mil83] D.L. Mills. DCN Local-Network Protocols. RFC 891 (Standard), December 1983.

[NB09] E. Nordmark and M. Bagnulo. Shim6: Level 3 Multihoming Shim Protocol for IPv6. RFC 5533 (Proposed Standard), June 2009.

[NNS98] T. Narten, E. Nordmark, and W. Simpson. Neighbor Discovery for IP Version 6 (IPv6). RFC 2461 (Draft Standard), December 1998. Obsoleted by RFC 4861, updated by RFC 4311.

[NNSS07] T. Narten, E. Nordmark, W. Simpson, and H. Soliman. Neighbor Discovery for IP version 6 (IPv6). RFC 4861 (Draft Standard), September 2007. Updated by RFC 5942.

[PHdVD08] C. Popoviciu, A. Hamza, G. Van de Velde, and D. Dugatkin. IPv6 Benchmarking Methodology for Network Interconnect Devices. RFC 5180 (Informational), May 2008.

[PJ99] C. Partridge and A. Jackson. IPv6 Router Alert Option. RFC 2711 (Proposed Standard), October 1999. Updated by RFC 6398.

[PJA11] C. Perkins, D. Johnson, and J. Arkko. Mobility Support in IPv6. RFC 6275 (Proposed Standard), July 2011.

[RMK+96] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear. Address Allocation for Private Internets. RFC 1918 (Best Current Practice), February 1996.

[SH04] P. Savola and B. Haberman. Embedding the Rendezvous Point (RP) Address in an IPv6 Multicast Address. RFC 3956 (Proposed Standard), November 2004.

[TGT08] F. Templin, T. Gleeson, and D. Thaler. Intra-Site Automatic Tunnel Addressing Protocol (ISATAP). RFC 5214 (Informational), March 2008.

[TT10] W. Townsley and O. Troan. IPv6 Rapid Deployment on IPv4 Infrastructures (6rd) – Protocol Specification. RFC 5969 (Proposed Standard), August 2010.

[TTP06] D. Thaler, M. Talwar, and C. Patel. Neighbor Discovery Proxies (ND Proxy). RFC 4389 (Experimental), April 2006.

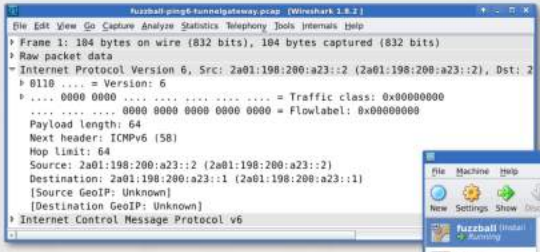
[Ull93] R. Ullmann. TP/IX: The Next Internet. RFC 1475 (Experimental), June 1993.

[VC04] R. Vida and L. Costa. Multicast Listener Discovery Version 2 (MLDv2) for IPv6. RFC 3810 (Proposed Standard), June 2004. Updated by RFC 4604.

[WKD+12] J. Weil, V. Kuarsingh, C. Donley, C. Liljenstolpe, and M. Azinger. IANA-Reserved IPv4 Prefix for Shared Address Space. RFC 6598 (Best Current Practice), April 2012.

Eine praktische Einführung in das Internet-Protokoll der Zukunft.

Dieser Workshop führt Sie durch die Installation, die Konfiguration und den Betrieb eines IPv6-Netzwerks. Erleben Sie selbst die Stärken und Schwächen von IPv6!
Lassen Sie sich mit praktischen Szenarien in Protokolle wie ICMPv6 und Neighbor Discovery einführen.



Paketanalysen
Tiefe Einblicke
in die Protokolle

Virtualisierung
Gefahrloses
Experimentieren

Lernen Sie Multicast und Übergangstechnologien kennen. Erfahren Sie, wie Sie ein Netzwerk allein mit IPv6 betreiben können, ohne auf IPv4 verzichten zu müssen. Wenden Sie das erworbene Wissen bei der Entwicklung eines Paketfilters unter Sicherheitsaspekten an.
Abschließend bereitet Sie der Workshop auf eine mögliche Migration im Unternehmen vor.

danrl.com